The Dissertation Committee for Amit Prakash

certifies that this is the approved version of the following dissertation:

Architectures and Algorithms for High Performance Switching

Committee:

Adnan Aziz, Supervisor

Scott Nettles

Yale Patt

Vijaya Ramachandran

Gustavo de Veciana

Harrick M. Vin

Architectures and Algorithms for High Performance Switching

by

Amit Prakash, B. Tech., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2004

To my parents

Acknowledgments

I would first like to thank my advisor, Prof. Adnan Aziz who has been a constant source of guidance and inspiration. He has not only been the best possible academic advisor, but also a mentor in all aspects of life.

I have been very fortunate to work with Prof. Vijaya Ramachandran on two papers about scheduling algorithms. I have learned a lot from her and I truly appreciate her support.

During my Ph.D. I have had frequent discussions with Prof. Gustavo de Veciana who has been extremely helpful and encouraging.

I appreciate all the advice and helpful feedback I got from Prof. Yale Patt, Prof. Scott Nettles, and Prof. Harrick Vin.

My stay in Austin has been one of the most memorable experiences of my life because of my friends, Aman, Gaurav, Jairam, Mitu, Praveen, Rashmi, Sadia, Shashank, Shimpa, Smitha, Sugat, Tameen, and Tanjeet. I had many fruitful technical discussions on various research problems with Sadia, Praveen, Sugat, Marghoob, Shashank, and Gaurav. Thank you for all your help.

Last, and certainly the most, I thank my parents, my wife, my brother, and my sister-in-law. Everyone has been extremely supportive, encouraging and understanding during the whole process. My wife Nidhi has helped me prepare so many presentations and documents over the time that by now she understands all that I have done in my Ph.D.

I thank you all for making this dream possible for me.

AMIT PRAKASH

The University of Texas at Austin

August 2004

Architectures and Algorithms for High Performance Switching

Publication No. _____

Amit Prakash, Ph.D.
The University of Texas at Austin, 2004

Supervisor: Adnan Aziz

Switches are ubiquitous in modern computing, appearing in wide-area networks, multiprocessor servers, and data storage systems. With the the advent of high-speed link technology, switches have become the bottleneck in moving data in the network. Existing switch architectures either require the interconnection network and packet buffers to work at a very high speed or require complex scheduling problems to be solved quickly. In this dissertation we investigate whether there are switch architectures that can support high-speed links that are simultaneously easy to schedule, and can be built out of inexpensive components.

The approach we take is using parallelism to solve complex scheduling problems. We choose switching architectures such that the corresponding scheduling problem can be efficiently solved with a reasonable amount of hardware. In particular, we present two switch architectures for which we have developed efficient scheduling algorithms. The first switch achieves optimum throughput and optimum average latency while the second switch guarantees optimum throughput only but uses considerably less hardware.

Contents

Chapter 1

Introduction

1.1 Motivation

Switches are devices that are used to implement networks—specifically, they store-and-forward discrete blocks of data from one point-to-point communication link to another. They play a critical role in modern computing of all forms, appearing in local- and wide-area networks, storage-area networks, next-generation buses, and servers based on shared-memory multiprocessors [23, 35, 16, 4, 14, 18, 48][20, Chapters 7.12, 8.12].

To be concrete, in this dissertation we will focus on switches that are used to build high-performance local- and wide-area networks; we note that many of our findings will be applicable to the more general domains noted above.

A switch used to be nothing more than a general-purpose computer connected via a standard bus to hardware for transmitting and receiving packets over links. Cisco routers in the early 1990s were built by adding network interface cards and control software to SUN workstations. It is no longer possible to implement the entire switch functionality using general purpose hardware for the following two reasons:

Mismatch between physical layer and switching technology On a relative basis, the performance of physical links has increased much more rapidly than the performance of

CMOS integrated circuits. For example, Ethernet LANs in 1990 operated at 10 Mbps; today, 10 Gig Ethernet is available. With DWDM technology, a single optical fiber will be able to carry over 1 Tbps [43, 45]. In contrast, in 1990 a state-of-the-art Intel 80386 was rated at 30 mips; today's fastest Intel P-IVs are rated at 2000 mips.

CMOS memories have increased in speed even more slowly, with DRAM cycle times decreasing from roughly 100 nanoseconds to 60 nanoseconds, and SRAM cycle times decreasing from roughly 40 nanoseconds to 2 nanoseconds.

Demands for greater functionality As networks become an integral part of society, additional features such as Quality of Service (QoS) and security are being expected of switches, and some of them require solving computationally challenging problems in a very short time.

As a result, in many networks today, switches, not links, are the bottleneck in moving data. Current high-end switches use brute-force solutions to such problems, e.g., by demultiplexing high-speed links and using replicated hardware. However, these approaches are *ad hoc*, expensive, and they do not scale well with increasing bandwidth and greater demands on QoS.

One can get an idea of just how difficult it is to design a high-performance switch by observing that the packet inter-arrival time on a switch with 40 Gigabit links and 512 bit packets is 13 nanoseconds. This corresponds to 39 instructions on a state-of-the-art 3.3 Ghz processor, rendering infeasible switch schedulers that solve complex graph algorithms (e.g., [30]) or make heavy use of pointer manipulation (e.g., [42]).

1.2 Background and definitions

Figure 1.1 shows the block-level architecture of a switch. *Input line cards (or input ports)* take packets from incoming links and compute the outgoing link to which the packet is to be forwarded. The *switch fabric* transfers the packets to the *output ports*, which transmit

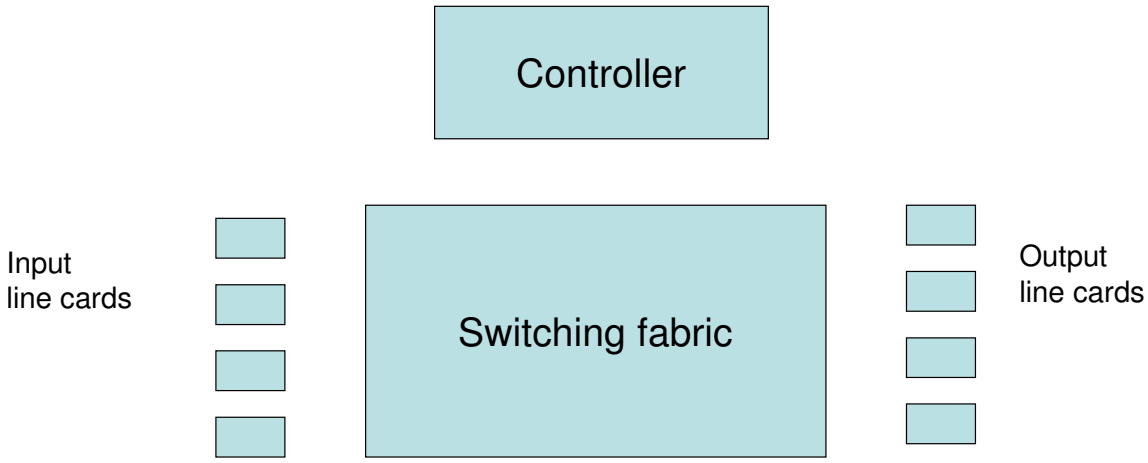


Figure 1.1: Architecture of a generic router

the packet on outgoing links. The buffering of packets can be performed at the input ports, switching fabric or the output ports.

We take the packets to be of fixed length (this may require segmenting packets at the input and reassembly at the output [23]). We assume that the switch has N input ports and N output ports with all links working at the same bandwidth (called line-rate). The *cycle time* of the switch is defined to be the amount of time it takes to transmit a packet through any of the links. The switch operates in cycles, i.e., at the beginning of each cycle a set of packets arrive at the input ports and at the end of each cycle a set of packets depart from the output ports. In each cycle at most one packet arrives at any input and at most one packet can be transmitted from any output.

1.2.1 Work conserving

A *work conserving* switch is the one in which a packet is always transmitted through an outgoing link if there is a packet available for transmitting. In other words, in a work conserving switch, a packet waits only if some other packet is being transmitted through the corresponding outgoing link. It is straightforward to show that for a given arrival sequence of packets, a work conserving switch achieves optimum average latency and throughput [23].

1.2.2 Admissible traffic

The rate at which a switch can receive and transmit packets is limited by the capacity of the links that carry those packets. Since multiple inputs can receive a packet for a particular output simultaneously, it is possible that an outgoing link becomes congested. If, for some time, packets arrive for a particular output at a higher rate, then the switch can buffer some of the packets and transmit them at a later time. However, if packets keep coming at high rate, eventually the buffers will fill up and packets will get dropped.

We define $a_{ij}(c)$ to be a sequence of random variables that takes value 1 if a packet arrives at the c-th cycle at input port i for output port j and zero otherwise. Define $I_i(c) =$

$\sum_j a_{ij}(c)$ to be the number of arrivals at an input port and $O_j(c) = \sum_i a_{ij}(c)$ to be the number of arrivals at an output port. Note that since length of a packet may not be multiple of cell size, the cell arrival rate may be more than the bandwidth divided by cell size. Let $A(c) = \langle a_{11}(c), a_{12}(c) \ldots a_{nn}(c) \rangle$. Let $\hat{A}(c) = \langle A(0), \ldots A(c) \rangle$.

Since we are dealing with infinitely long sequences we will need mathematically precise definition of admissible traffic. Different researchers have used different models of admissible traffic. Most earlier work on switch scheduling has been on Markovian arrival process in which the average cell arrival rate for any input or for any output is less than one. We would also like to consider adversarial traffic, when the arrival sequence under the constraint that there exists an integer $T > 0$ and a real number $\epsilon \in (0, 1)$ such that during any consecutive T cycles the number of arrivals for any input or any output is no more than $\epsilon(T-1)$. In this dissertation we use the following definition of admissible traffic which includes both the adversarial traffic patterns as well as rate limited Markovian traffic patterns.

Definition 1 For an integer T and a real number ϵ, an arrival process is defined to be (T, ϵ)-*admissible* if for all i, $\mathrm{E}[\sum_{t=u}^{T+u} I_i(t) | \hat{A}(t-1)] \leq T(1-\epsilon)$ and for all j, $\mathrm{E}[\sum_{t=u}^{T+u} O_j(t) | \hat{A}(t-1)] \leq T(1-\epsilon)$.

In general, an arrival process is called admissible if there exist some integer $T > 0$ and real number $\epsilon \in (0, 1)$ such that the arrival process is (T, ϵ)-*admissible*.

1.2.3 Stability and 100% throughput

Roughly, throughput is defined to be the limit over time of the ratio of number of packets that have been forwarded by the switch to the number of packets that have arrived at the switch.

We define switch to be *stable* (in the bounded input bounded output sense) if under admissible load, the total number of packets in the system never diverges to infinity.

Note that stability implies 100% throughput for admissible load but the converse is not true. Specifically if the queue length grows as a sublinear function of time, then we get 100% throughput (assuming the number of packets served grows linearly in time) but not stability.

Stability is a weaker condition than work conserving. However, for a switch that is not work conserving, it is highly desirable that it be stable. Stability guarantees that if the switch is run for large number of cycles under admissible load, except for a finite number of packets, all the packets will be transferred. Thus, asymptotically, the switch achieves 100% throughput.

In order to achieve desired performance some switches use faster interconnections than the line-rate. The ratio of speed of the interconnection network inside the switch to the line-rate is called *speed-ratio*.

1.2.4 Order preserving

A switch is called *order preserving* if any two packets that belong to same transport layer flow, depart in the order they arrive. When the flow information is not available, it is safe to assume that all the packets that arrive at a certain input and depart through certain output belong to a single flow.

1.3 Ideal switch

Now we are ready to define an *ideal switch*. The properties desirable of a switch are following:

- **High throughput**: Ideally, the switch should be stable for any admissible traffic pattern.
- **Low latency**: A work conserving switch has optimum average latency. Thus, a work conserving switch would be ideal in terms of latency. However, in many cases where

a switch is expected to provide QoS guarantees to specific flows, it is desirable that scheduler be able to control the order in which different flows get serviced.

- **Large packet buffer**: Switches are required to have large packet buffers to deal with congestion. It is standard practice in the Internet to provide for 0.5 second of buffering; thus a line card supporting 40 Gbps link would need 2.5 GB of buffer space. As the buffer needs to support high data-rate, often it is implemented using expensive SRAMs. Most crossbar-based architectures require static partitioning of available buffer space between different inputs and outputs. Thus, it can happen that some of the buffers are empty while others are full, causing packet drops. The total available memory can be used in a better fashion if it is shared across all ports. Some of our work is motivated by this fact.

- **Order preserving**: If the transport layer protocol does have provision for reordering packets that arrive out of order, packets arriving out of order become useless. Although many protocols, such as TCP, can take care of packet reordering, it is standard practice to use order preserving switches in the Internet so that bad implementations of TCP and other protocols such as UDP do not suffer.

1.4 Previous work

Early switches were built by connecting all the line cards to a single shared memory through a bus. As packets arrived at the inputs they were stored in the shared memory through the bus and when a packet was scheduled to depart it was read from the memory and written in the corresponding line card. This switch had all the characteristics of an ideal switch but it required each packet to cross the bus and memory interface twice. Thus, the bus bandwidth and the memory bandwidth must be $2N\times$ of the line-speed. For modern high-speed switches this is infeasible.

Modern high-speed switches use crossbar or similar interconnection networks to

connect input ports and output ports. Conceptually, a $N \times M$ crossbar has N inputs, connected to each of M outputs through a matrix of $N \cdot M$ switches, as described in Figure 1.2. By programming these switches, a crossbar can realize any mapping of inputs to outputs. Since the size of a crossbar increases as $\Theta(N^2)$ for an $N \times N$ switch, it is not possible to make very large crossbars. Many hierarchical interconnection networks such as Benes or Torus networks have been studied [14] to alleviate this problem. In this dissertation we will work with crossbar as an interconnection network, although it can be replaced with a hierarchical interconnection network.

The two classic crossbar-based architectures are input-queued (IQ) architecture and output-queued (OQ) architecture. As the names suggest, in an IQ switch the packets are buffered only at the input ports and in an OQ switch packets are buffered only at the output ports. A purely OQ switch has all the great qualities of an ideal switch except for memory utilization, but it needs the crossbar to run at speed-ratio of N and each memory must support N write and one read every cycle; OQ architecture is not scalable with number of ports. IQ architecture can work without any speed-ratio but it requires very complex scheduler to guaranty good throughput [30]. Some researchers have studied switches with some constant speed-ratio; they require buffering at both input ports as well as output ports. The time complexity of even these switches is formidable.

1.4.1 Input queued switches

An input queued switch in which the memories and interconnects work at line-rate, at most one packet can be transfered from any input and at most one packet can be transfered to any output. Hence, the scheduling problem for IQ switch can be cast in terms of computing a matching in a bipartite graph. Specifically, the schedule corresponds to a matching in the graph whose vertices are the input ports and output ports, with edges connecting an input port to an output port when a packet resides at the input port for that output port.

It is known that stability can be achieved by solving a maximum weighted matching

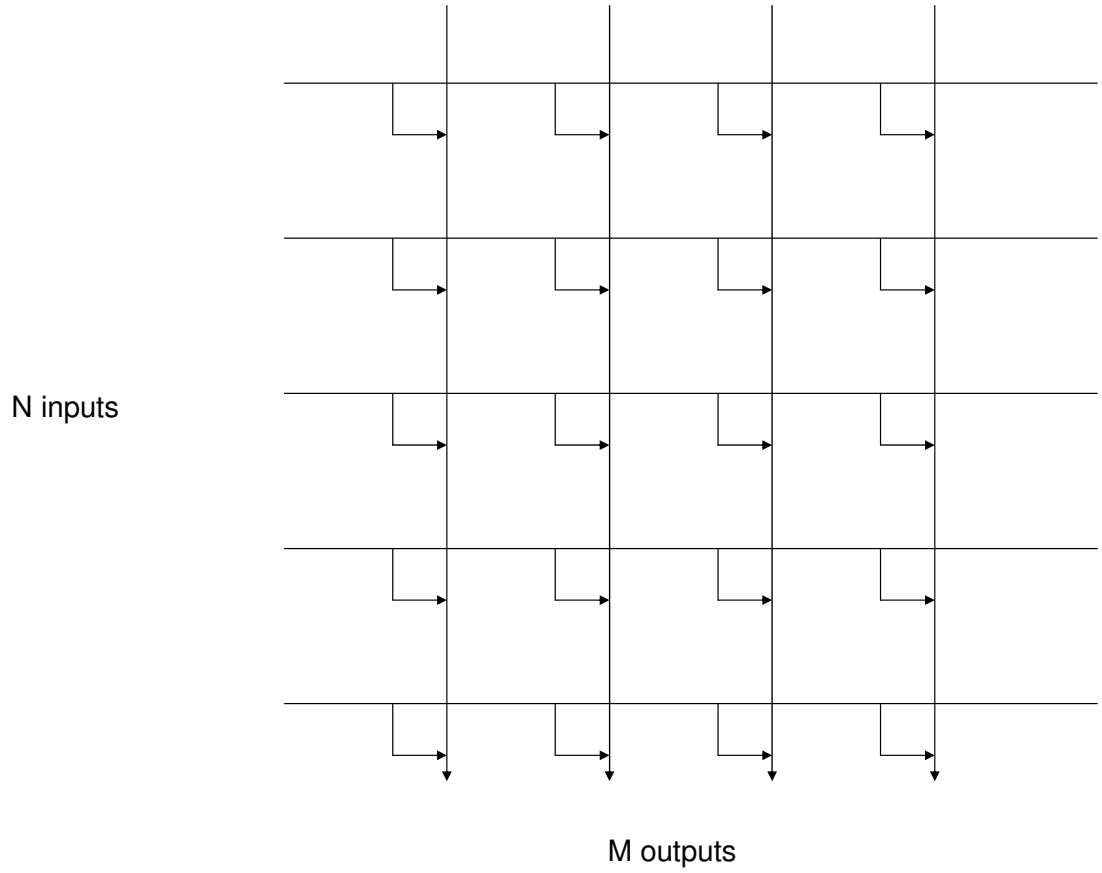


Figure 1.2: An N input M output crossbar

problem in this graph if the arrival process is Bernoulli, while maximal matchings can be unstable [29]. There exists a randomized parallel algorithm for the weighted matching problem [25] that is efficient in the sense that it runs in $O(\lg^4 N)$ time for a graph with N vertices. However, the hidden constants are too large and it is not practical because of its hardware cost and runtime for reasonable values of N.

Tassiulas [46] proposed the idea of selecting a matching uniformly at random from the space of all the matches and replacing it with the matching used in the previous cycle only if it has a higher weight. The main idea is that once the queues become large, the weights in the graph are going to change very slowly. Hence, eventually, the randomly chosen match will correspond to a high weight match and it will continue to be used until a significant amount of packets in the queues have been discharged. Although this algorithm has very low complexity and in theory it achieves stability, but since the space of all matches is so large ($n!$ to be precise) that it takes an inordinately long time for the random permutation generator to come up with the right permutation. Hence, the queue lengths can be quite large. Giaccone *et al.* [19] propose several heuristic approaches to overcome this problem by using the information in current arrivals or considering some random neighbors of current match. These approaches do show good improvements from the original algorithm in simulation. However, it is not clear that what theoretical guarantees can be made about improvement. Furthermore, all the approaches in [19] require a significant amount of extra hardware for computing the weight of each match and comparing them to get the match with highest weight.

Dai and Prabhakar [13] have shown that using maximal matches for scheduling in conjunction with speed-ratio of 2 leads to stability for a wide class of traffic; however the best deterministic parallel algorithm for maximal matching [21] is also not very practical.

Anderson *et al.* [2] introduced Parallel Iterative Matching (PIM), a randomized algorithm for computing maximal matchings. PIM iteratively pairs up unmatched vertices, and is very simple to implement. It computes a maximal match with high probability in

$O(\lg N)$ iterations. Since it computes a maximal matching, it can be used with speed-ratio of 2 in an IQ switch. McKeown [28] proposed iSLIP algorithm, that can construct maximal matches; it uses rotating priorities in stead of random bits to choose between different pairings in each iteration. This eliminates the need for random number generators but the worst case requires N iterations.

1.4.2 Emulating output-queued switches

Chuang *et al.* [10] have studied the possibility of getting the performance of an OQ switch, specifically optimum latency and throughput, using little or no speed-ratio in a crossbar. They have shown that a switch with buffering at both the input and output ports can emulate an OQ switch by performing 2 reads and 2 writes on the input and output buffers, respectively, and running the switch fabric at speed-ratio of 2. They have further shown that no scheduling algorithm exists for speed-ratio less than $2 - 1/N$. Their approach hinges on a sophisticated scheduling algorithm which solves an instance of the stable marriage problem. The best known parallel algorithm [17] for solving the stable marriage problem has complexity $\Omega(\sqrt{N} \cdot \lg^3 N)$ using $\Omega(N^4)$ processors, and is based on interior point methods for linear programming; it is impractical to implement in hardware.

1.4.3 Other architectures

Load balanced Birkhoff-von Neumann switch

Chuang *et al.* [8, 9] proposed Birkhoff-von Neumann switch that achieves 100% throughput with O(1) work if the arrival rates are known apriori and it is admissible. The main idea is that, given the arrival rates, a set of permutation matrices $P_1 \ldots P_k$ and corresponding weights $w_1 \ldots w_k$ can be computed such that, if each permutation P_i is realized w_i fraction of the cycles, then service rate is guaranteed to be not less than arrival rate. Since the computation of schedule can be done apriori, in each cycle only constant amount of work is needed.

However, in most cases the arrival rates are not known in advance. Chuang *et al.* further propose a two stage switch in which the first stage has a crossbar that transfers arriving packets into an intermediate set of buffers and the second stage just reads the packets from these buffers to transfer them to corresponding outputs. Both the crossbars realize a fixed set of permutations in a round robin fashion. This again yields 100% throughput, however, this algorithm is not order preserving. Keslassy and McKeown [24] propose to use a resequencing buffer at the output to preserve the packet order. They show that an $\Theta(N^2)$ packet buffer is required and packets may get delayed by $\Theta(N^2)$ cycles because of this.

Buffered crosspoint switch

Buffered cross-point switch is an interesting architecture that has a packet buffer for each input-output pair [49]. This architecture can achieve performance of an OQ without any speed-ratio and a very simple scheduler. However, it requires N^2 buffers. Partitioning the available memory into N^2 buffers makes it very inefficient in terms of buffering. Even under moderate load, with a good probability one of the buffers will be full and cause packet drops. Furthermore, the cost of peripheral logic used around memories increases as N^2 in this architecture which makes it impractical for big switches.

1.5 Key contributions

In this dissertation we present architectures and parallel algorithms for building high-speed routers. The results presented in this dissertation are interesting from both theoretical and practical view points. While we improve asymptotic bounds on scheduling, we also present numerical analysis and simulations to show that the results are useful in practice.

We formalize the switch memory switch (SMS) architecture and demonstrate that it can emulate output queuing with speed ratio of one [42]. We present a randomized algorithm for scheduling a router using SMS architecture that runs in $O(\log^* N)$ time [3]. We

further improve this algorithm by use of pipelining [39]. We have shown that our algorithms are near optimal in time complexity and use of memory, and optimum in throughput and average latency.

We present a simple randomized scheduling algorithm for achieving 100% throughput in input queued switches [34]. Although this algorithm does not achieve optimum latency it is an extremely simple scheduling algorithm with minimal hardware cost.

I greatly appreciate contributions from my co-authors, Prof. Adnan Aziz ([42, 3, 39, 34]), Rina Panigrahy ([34]), Prof. Vijaya Ramachandran ([3, 39]), and Sadia Sharif ([42]). This work has been supported by grants from NSF, state of Texas higher education council, and gifts from IBM and Synopsys.

1.6 Other problems in building high-speed networks

This dissertation is focused on scheduling of packets in high-speed networks. There are other computationally intensive tasks that a switch needs to perform, such as admission control, packet classification, and error correction and detection.

In an IP network, when a packet arrives at a switch, at the very minimum the switch needs to look at the destination IP address of the packet and determine where it needs to be forwarded. Often switches look at other fields such as source IP address, source and destination ports to determine what action needs to be taken. These actions are taken based on a prioritized set of rules specified by the system administrator. Packet classification is difficult because the time budget for doing it is usually very small and the rule-sets are very large (of the order of thousands).

Most wired networks have only error detection capability implemented through Cyclic Redundancy Code (CRC). However wireless networks heavily depend upon the error correcting codes. Error correction and detection schemes are also interesting for packet switching. If a scheduling algorithm does not guaranty delivery of all the packets to the output, but delivers most of the packets, it can be viewed as an erasure channel. Thus,

packets can be encoded into smaller blocks of data such that even if some of these blocks are missing the packet can be reconstructed. Such an approach will also depend upon fast encoding and decoding schemes.

Low Density Parity Check (LDPC) codes are one of the leading codes used for error correction. However an LDPC decoder requires significant amount of computation. A direct implementation of LDPC codes in hardware is expensive and dominated by interconnects [5].

We have made contributions in the field of packet classification [37, 38, 40, 41] and implementation of LDPC decoders in hardware [32]. Since the focus of this dissertation is on scheduling algorithms for switching packets, we will not go into further detail of our work on the above mentioned problems in this dissertation.

Chapter 2

Randomized Scheduling Algorithm for SMS Architecture

In this chapter we introduce the *switch-memory-switch (SMS)* architecture for packet switching. There are three main advantages of using an SMS architecture over other architectures: **(1)** the average delay can be minimized (as in output queuing [23]), **(2)** the buffer memories need to support only one read and one write per cycle (as in input queuing), and **(3)** with a good scheduling algorithm, the packets can be distributed almost equally among the buffer memories to make sure that a packet gets dropped only if all the buffers are full (thus, the same packet drop rate can be achieved with smaller memories as compared to an output-queued or input-queued switch).

The SMS architecture buffers packets in small memories placed between the input and output ports. In this architecture, the output ports have buffers that need to hold just one packet, and the input ports have buffers of small size. Thus, the main buffers in this architecture are the small memories placed between inputs and outputs, which operate together.

As depicted in the Figure 2.1, the N input ports are connected to write ports of M memories through an $N \times M$ crossbar, and the N output ports are connected to the read

ports through another $N \times M$ crossbar. As the packets arrive, they are transferred to one of the memory banks through the first crossbar and when it is time for departure of a packet, it is read from the memory bank and transferred to the corresponding output port through the second crossbar.

This is the architecture used by some of the fastest routers available today, such as the M160 Internet core routers from Juniper Networks [33].

In our work, we have considered a more general model where each buffer can support s reads and one write. In most designs s would be 1. However, we can trade speed for reducing number of memory banks by increasing s. We do not consider s writes and s reads as the analysis would be identical to using $s \cdot M$ memories that support one read and one write per cycle.

2.1 Emulating output queuing

An OQ switch is ideal in terms of throughput as well as average latency. Since output-queuing is highly desirable, our goal is to emulate the behavior of an $N \times N$ output-queued switch that has buffer memory space for L packets at each output using an SMS architecture. By emulation, we mean that for any arrival sequence **(1)** a packet is dropped by the SMS router iff it will be dropped by the output-queued router, and **(2)** if a packet is not dropped then the cycle in which it departs the SMS router must be same as the cycle in which it would have departed the output-queued router.

The cycle in which a packet would have departed an output-queued router is referred to as its *time-stamp*. When a packet arrives at an input of an SMS router, its time-stamp is computed as described in Section 2.3.1. In each cycle, packets at the inputs are written to a subset of memories through the first interconnect, and packets whose time-stamp is equal to the current time are read from the memories and transferred to the outputs through the second interconnect.

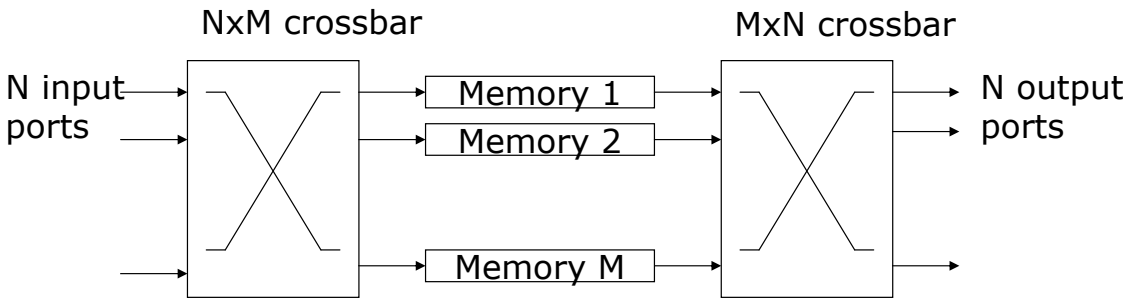


Figure 2.1: Switch Memory Switch architecture

2.2 Conflicts

In the SMS architecture, each memory can support one read and s writes per cycle. Hence, packets cannot be arbitrarily placed in the memories. A packet faces two kinds of conflicts. More than s packets that arrive at the same time cannot be written to the same memory; this is referred to as an *arrival conflict*. Since there are N input ports, the maximum number of arrival conflicts a packet can have is $\lceil (N-1)/s \rceil$. *Departure conflicts* occur if multiple packets in the same memory need to depart simultaneously through different outputs. Since there are N outputs, a packet can have departure conflicts with at most $N-1$ memories. Hence, if the number of memories is $M \geq \lceil (N-1)/s \rceil + N$ there will always be a conflict-free memory for each packet. A conflict-free memory for an input is said to be *compatible* with that input.

2.3 Scheduler tasks

In order to construct a conflict-free schedule for transfer of packets, the scheduler has three tasks to perform in every cycle.

Task 1 Compute the time-stamp of all the newly arrived packets.

Task 2 Match the newly arrived packets to memories such that there are no departure and arrival conflicts.

Task 3 Read packets whose time-stamp is equal to the current time and transfer them to the output.

Since the time-stamp of a packet is known when it is written to a memory, Task 3 is simple. We briefly describe how Tasks 1 and 2 are performed. Task 2 is the most complex step and is the focus of our work.

2.3.1 Task 1: Time-stamp computation

An array $E[1 \ldots N]$ stores the earliest available time-slot for each output.

Let P_1^o through $P_{c^o}^o$ be the packets destined for output port o that arrived in the cycle T and let them be ordered according to the id of the input port on which they arrived. Then the time-stamp of packet P_i^o is set to $(E[o] + i)$ and $E[o]$ is set to $\max(E[o] + c^o, T)$. This time-stamp assignment is consistent with the requirement of emulating an output-queued router, and can be efficiently computed by a prefix-sum computation.

If the difference in time-stamp of a packet and current time is greater than L then it is dropped. This behavior is consistent with the behavior of an output-queued router with buffer of size L at each output.

2.3.2 Task 2: Scheduling using graph matching

For routers that are relatively small and slow, the SMS architecture can emulate output-queuing by using a straightforward greedy sequential algorithm to compute an assignment of incoming packets to compatible memories. However, for routers with many ports operating at high speeds, the sequential algorithm is not fast enough to compute the assignment.

The only known parallel algorithm for computing the assignment is that of Prakash, Sharif, and Aziz [42]; however, it uses a graph coloring algorithm that requires building and manipulating complex data structures. Subsequent to our work, Iyer *et al.* also presented an architecture isomorphic to SMS, however, they only consider a sequential scheduling algorithm that has a running time of $\Omega(N)$ [22].

2.4 Randomized parallel scheduler for SMS

Here we present a randomized algorithm for performing Task 2. We call this algorithm *Randomized Parallel Switch Scheduler* (RiPSS). In this section each input is identified with the packet that just arrived at that input. Recall that an input i is *compatible* with a mem-

ory m if the packet that just arrived at i can be stored in memory m without *arrival* and *departure conflicts* (see Section 2.2).

At the beginning of a cycle, the time-stamp of each input port is broadcast to each memory and memories construct a list of inputs that are compatible with the memory. The algorithm then works in rounds according to the 'Basic Matching Process' (BMP) given below. Anderson *et al.* [2] proposed a similar algorithm called Parallel Iterative Matching (PIM) for a completely different architecture, namely a crossbar-based input-queued router with "virtual output queues." In their case they need to compute a maximal matching in an arbitrary bipartite graph, and they prove that the expected number of rounds for their algorithm is $O(\log N)$.

Initially all the memory banks are unmatched.

Basic Matching Process:

(1) In parallel each unmatched memory sends a message to a random compatible input port.

(2) In parallel each input port i picks a memory bank j that sends it a message and assigns its current packet to that memory bank. It then broadcasts a bit to all memory banks to inform them that it is no longer available to be matched (the bit sent to memory bank j is a 1 and the bit sent to all other processors is 0).

(3) In parallel each memory bank that receives a 1-bit from its matched input decrements a counter initially set to s. If the counter goes down to zero, the processor declares itself matched.

2.4.1 Analysis of RiPSS

In this section we establish that if $M = (N + \lceil N/s \rceil + \epsilon N)$, for any $\epsilon > 1/2^{\log^* N}$, then w.h.p. in N, the number of rounds needed to match every input to a compatible memory bank is $O(\log^* N)$. The analysis views the computation in the 'balls-in-bins' framework, and the slight excess in the number of available memory banks over the bound of $(N + \lceil (N-1)/s \rceil)$ given in section 2.2 allows for the acceleration in the matching pro-

cess in successive rounds leading to the $O(\log^* N)$ bound. Randomized strategies with $O(\log^* N)$ complexity are known in the literature for other scenarios, e.g., in the context of highly-parallel algorithms for the CRCW PRAM [27] and in emulating shared-memory on distributed memory (see, e.g., [12]), and our strategy is similar to these in terms of accelerating progress in successive rounds, although the exact method and analysis are different.

Lemma 2.4.1 If there are k unmatched inputs at a beginning of a round then there must be $(\epsilon N + \lceil k/s \rceil)$ unmatched compatible memories for each input.

Define a round that starts with k unmatched inputs to be *successful* if it ends with at most $ke^{-(1/s+\epsilon N/k)} + \sqrt{2M \log M}$ unmatched inputs. In the following lemma we prove that w.h.p. a round is successful.

Lemma 2.4.2 If there are k unmatched inputs and M memories at the beginning of a round and each input can be matched to at least $\epsilon N + \lceil k/s \rceil$ memories, then the expected number of unmatched inputs at the end of that round is at most $ke^{-(1/s+\epsilon N/k)}$. Furthermore the probability that the number of unmatched inputs exceeds its mean by more than $\sqrt{2M \log M}$ is at most $\frac{1}{M}$.

Proof: The bound on the expectation is straightforward, and the high probability bound is obtained using Azuma's inequality on a suitable martingale. The proof is in the Appendix. ■

Since $1/M \leq 1/N$, the first $O(\log^* N)$ rounds are successful w.h.p. in N. The following discussion assumes that they are successful.

Let k_r be the number of unmatched inputs at the beginning of round r. We know that $k_0 = N$ and k_r decreases in successive rounds. Let R be the last round for which $k_R \geq W\sqrt{2M \log M}$, where W is a constant chosen to ensure that $k_{r+1} \leq (k_r/\alpha)e^{-\frac{\epsilon N}{k_r}}$ for $r < R$, where $1 < \alpha < e^{1/s}$. We will prove that $R = O(\log^* N)$. (Note that by Lemma 2.4.1 and a Chernoff bound, w.h.p. in N all inputs are matched in round $R + 1$). To state

the following lemma we first define the double arrow notation. Let $(a \uparrow\uparrow 1) = a$ and, $(a \uparrow\uparrow b+1) = a^{(a\uparrow\uparrow b)}$. In other words, $(a \uparrow\uparrow b)$ is a b high exponential tower of a.

Lemma 2.4.3 For every constant $c > 0$ there exists a constant $b = e^{\epsilon/c}$ such that if there are k unmatched packets at the beginning of a round $r < R$ and for some positive integer i we have $k \leq \frac{cN}{b\uparrow\uparrow i}$ then the number of unmatched inputs at the end of that round is at most $\frac{k}{\alpha(b\uparrow\uparrow(i+1))}$, w.h.p. in N.

Proof: The number of unmatched inputs at the end of round is at most $\frac{k}{\alpha e^{\epsilon N/k}} \leq \frac{k}{\alpha e^{\epsilon(b\uparrow\uparrow i)/c}} = \frac{k}{\alpha b^{(b\uparrow\uparrow i)}}$. ■

>From Lemma 2.4.3 it trivially follows that $k_{r+1} \leq k_r/\alpha$. Let $A = \lceil \log_\alpha \frac{\ln 2}{\epsilon} \rceil$. Hence, after A initial rounds we have $k_A \leq N\epsilon/\ln 2$. Now substituting $c = \epsilon/\ln 2$ in Lemma 2.4.3 we have $b = 2$, and hence, $k_r \leq \frac{N\epsilon}{(\ln 2)(2\uparrow\uparrow i)}$ implies $k_{r+1} \leq \frac{k_r}{\alpha(2\uparrow\uparrow(i+1))} \leq \frac{N}{(2\uparrow\uparrow(i+1))}$.

Since $k_A \leq N\epsilon/\ln 2$, applying the above inequality repeatedly we obtain $k_{r+A} \leq \frac{N}{\alpha(2\uparrow\uparrow r)}$. Thus, at the end of $A + \log^* N$ rounds we cannot have more than $W\sqrt{2M \log M}$ unmatched inputs. Since $W\sqrt{2M \log M}$ inputs can be matched in a single round w.h.p. in N, we can match all the inputs in $A + \log^* N + 1 = O(\log^* N)$ rounds, if $\epsilon = \Omega(1/2^{(1/s)\log^* N})$. This gives us the following theorem.

Theorem 2.4.4 If the router can transfer s packets to each memory bank in a cycle, then if $M = (N + \lceil N/s \rceil + \epsilon N)$, repeated applications of the BMP will match all inputs to memories in $O(\log^* N)$ rounds w.h.p. in N, if $\epsilon = \Omega(1/2^{(1/s)\log^* N})$.

2.4.2 Worst case bounds on performance

The theorems presented in Section 2.4.1 tell us only the asymptotic behavior of RiPSS. In particular they do not tell us what the hidden constants are and for what value of N (the number of input ports) and M (the number of memory banks in the SMS architecture) we obtain acceptably small probability of failure. Since we do not have simple closed form

expressions for the exact probability of failure and for the number of rounds, in this section we present concrete upper bounds on the probability of failure and number of rounds needed to limit probability of failure to a certain value.

In a given cycle, we know that each input can be incompatible with at most $N-1$ memories. Thus, each input must have at least $M-N+1$ compatible memories. We consider the *worst case* scenario, where each input has exactly $M-N+1$ compatible memories and all inputs contend for the same set of $M-N+1$ memories. Thus, the compatibility graph would be a complete $N \times (M-N+1)$ bipartite graph. Let $P_m(i,j,k)$ be the probability of the event that if k balls are thrown uniformly at random into j bins then there are exactly i non-empty bins. Thus, probability of i packets being matched in such a scenario would be $P_m(i, N, M-N+1)$. For this to happen, if the first $k-1$ balls fall into exactly i bins then the last ball must also fall in one of these i bins. Alternately, if they fall in $i-1$ bins then the last one must fall in a new bin. This gives us the following recurrence relation,

$$P_m(i,j,k) = P_m(i,j,k-1) \cdot \frac{i}{j} + P_m(i-1,j,k-1) \cdot \frac{j-i+1}{j}$$

Now let the probability $P_r(n,m,r)$ be the probability of matching n packets to a subset of m memories in r rounds when each input of the n inputs are compatible with each of the m memories. Thus,

$$P_r(n,m,r) = \sum_{i=1}^{n} P_m(i,n,m) \cdot P_r(n-i, m-i, r-1)$$

A similar approach can be used to computing the expected number of rounds. We emphasize that the solution to the recurrence relations provides only an upper bound, since the relations assume a very pessimistic scenario.

Table 2.1 shows the minimum number of rounds needed to ensure that the probability of all packets being matched is at least 0.999. The numbers in table show that, if $\epsilon \geq 0.5$ we never need more than 3 rounds.

ϵ \ N	16	32	64	128	256	512	1024	2048	4096
0.1	4	4	4	5	5	5	5	5	5
0.5	3	3	3	3	3	3	3	3	3
1.0	3	3	3	3	3	3	3	3	3

Table 2.1: Upper bound on number of rounds needed to match all the packets with probability ≥ 0.999 in a switch with N inputs and $(2+\epsilon)N$ memories (computed from the recurrence relation **for worst case traffic**).

Figure 2.2 depicts the expected number of rounds for various values of N and M. It can be seen from the figure that even for the case where $M = \lfloor 2.1N \rfloor$ (i.e., $\epsilon = 0.1$) we need less than 4.01 rounds for a 4096 port switch on an average. With $N = 3N$ the expected number of rounds remains below 2.02. In Figure 2.3 we plot the probability of matching all inputs at the end of a fixed number of rounds. With $M = \lfloor 2.1N \rfloor$ and 4 rounds, the probability of matching all inputs is very close to one for a switch with up to 4096 ports. In Figure 2.4 we look at the effect of increasing the number of memory banks on the expected number of rounds. We use $M = \lfloor (2+\epsilon)N \rfloor$ and plot the expected number of rounds for different values of N and ϵ. Obviously, as ϵ increases the expected number of rounds decreases. However, there does not seem to be much gain after $\epsilon = 0.5$.

2.4.3 Simulation studies on stochastic traffic

The failure probabilities and expected number of rounds computed in the previous section provide upper bounds on worst case traffic. However, we do not know of any explicit arrival sequence that would achieve those bounds and it may be the case that no such sequence exists. Similarly, we do not know whether the bound on M provided in Theorem 2.5.5 is tight or not. In this section we present results of simulation of SMS switch for different values of N and M and packet arrival patterns. Even though Lemma 2.5.4 guarantees the existence of a perfect match only when $M \geq 2N - 1$, in our simulations we observed that

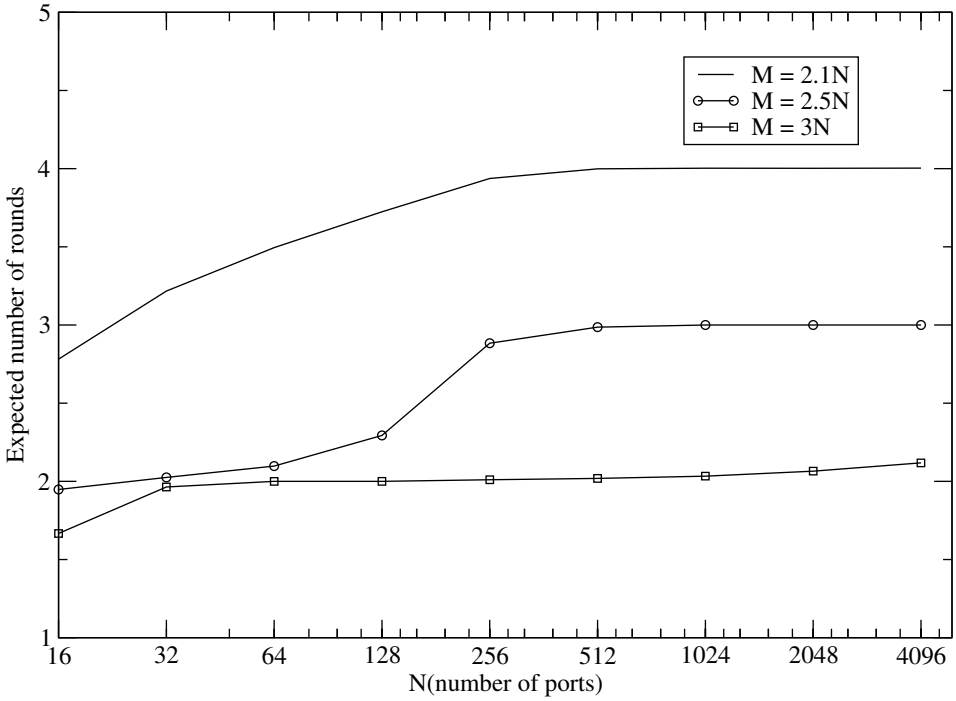


Figure 2.2: Bound on the expected number of rounds needed to match all the packets as a function of N (obtained from the recurrence relation **for worst case traffic**)

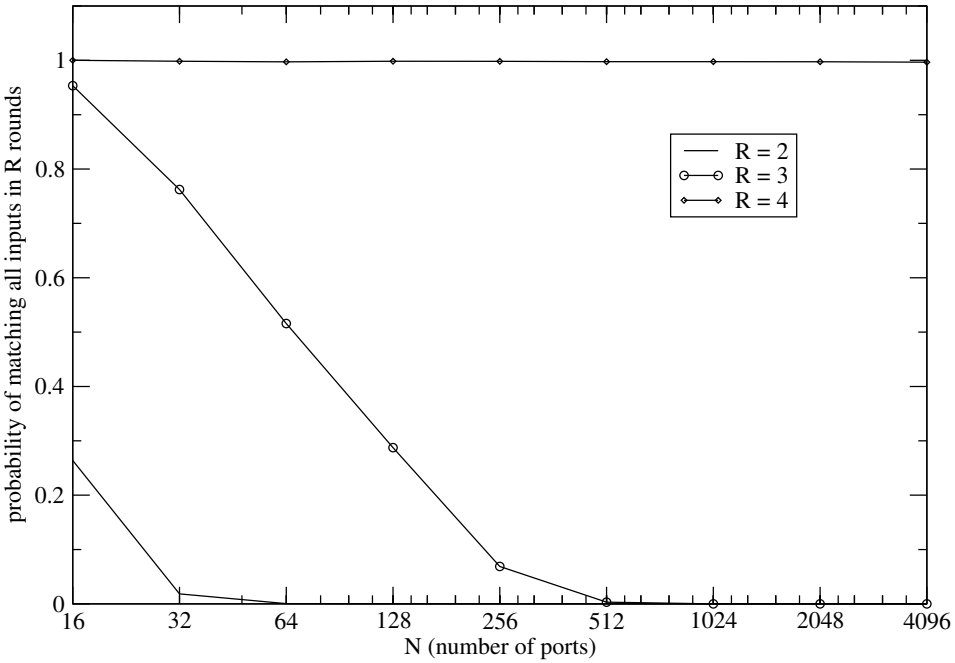


Figure 2.3: Lower bound on probability of matching all inputs in R rounds (obtained from the recurrence relations **for worst case traffic**)

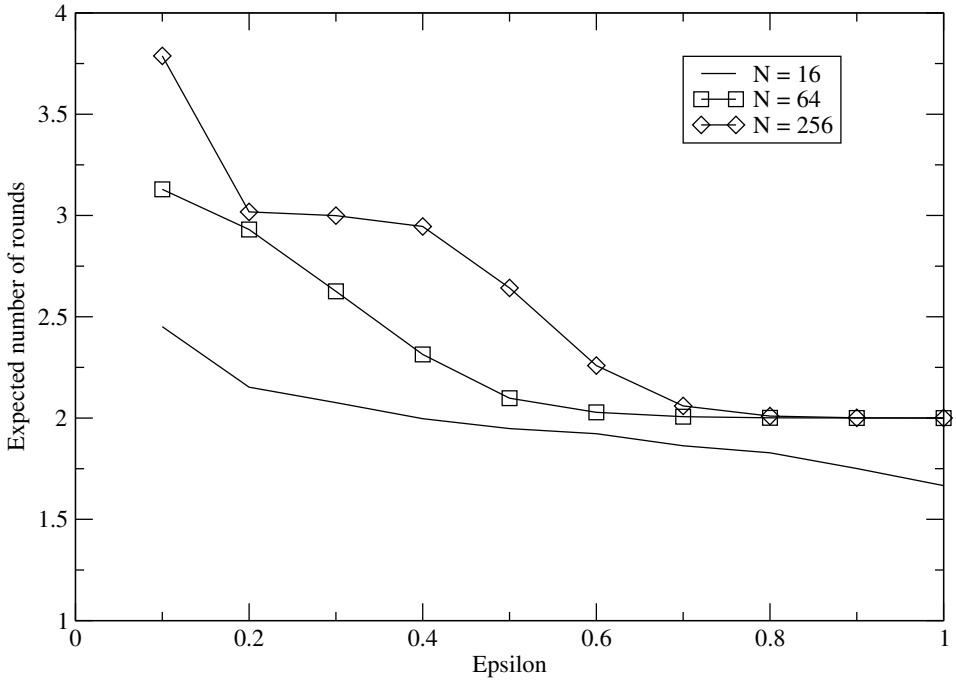


Figure 2.4: Upper bound on average number of rounds needed to match all the packets as a function of ϵ, where $M = \lfloor (2+\epsilon)N \rfloor$ (obtained from the recurrence relations **for worst case traffic**)

RiPSS never failed to match all the inputs if $M \geq 1.6N$ (with $M = 1.5N$ we saw failures in 0.1% of the cycles).

Figure 2.5, Figure 2.6, and Table 2.2 refer to the results obtained by simulating uniform Bernoulli arrival, in contrast to the worst case arrivals assumed in the previous section. Figure 2.5 plots the average number of rounds needed to match all inputs with uniform Bernoulli traffic; as might be expected, the average number of rounds needed here is less than the worst-case upper bound in the previous section. Even when only $1.6N$ memories are used, the packets are matched in less than 3.1 rounds on an average for N up to 4096. Figure 2.6 shows the number of rounds needed for different values of M/N.

Table 2.2 shows the number of rounds needed to bound the probability of failing to match all inputs to at most 0.1%, for different values of N and M, as observed by simulating the system for 100,000 cycles. Even with $1.6N$ memories, all packets are matched 99.9% of the time if only 3 rounds are used. Further, in a cycle in which the scheduler does not construct a perfect match, only a small number of packets remain unmatched, hence only a small number of packets are dropped.

c \ N	16	32	64	128	256	512	1024	2048	4096
1.6	3	3	3	3	3	3	3	3	3
2.0	3	3	3	3	3	3	3	3	3
2.5	2	2	2	2	2	2	2	2	2

Table 2.2: Number of rounds needed to match all the packets more than 0.999 of the times, in a switch simulated for 100,000 cycles with N-inputs and $c \cdot N$ memories for **uniform Bernoulli traffic**)

2.5 Optimum use of memory in SMS

The memories used to buffer packets contribute significantly to the total cost of a router. Thus, it is important to minimize both the number of memories used, and the size of each

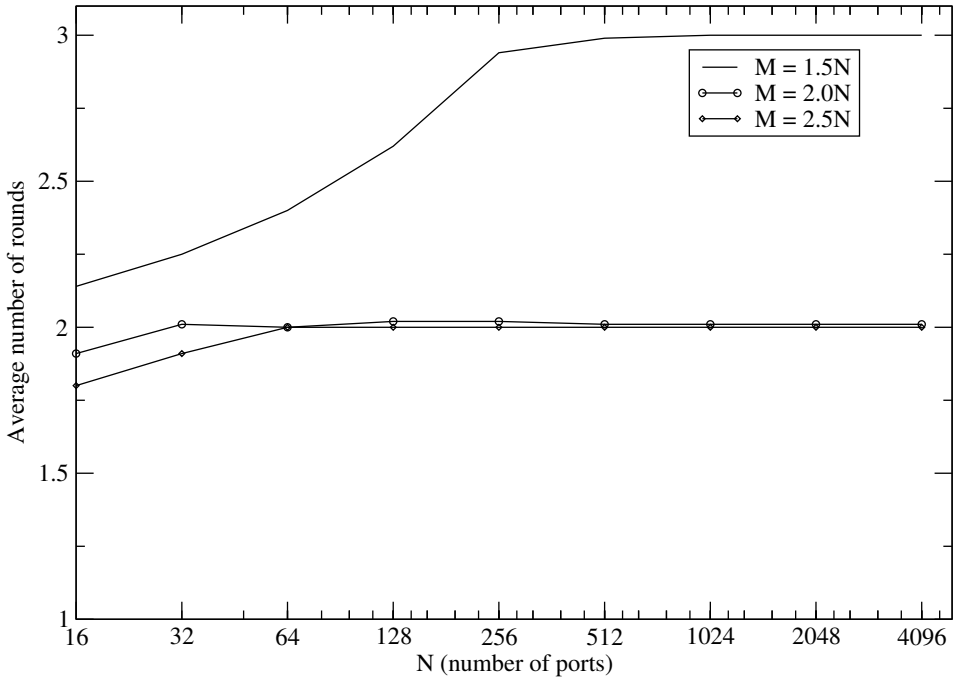


Figure 2.5: The average number of rounds needed to match all packets as function of number of ports (as observed in simulations with **uniform Bernoulli traffic**, simulated for 100,000 cycles)

memory.

Routers need a large amount of memory in order to achieve low drop rates. Studies of Eckberg *et al.* [15] reveal that packet drop probability significantly decreases if memories can be shared across the queues for different outputs. Eckberg *et al.* show that for a Poisson packet arrival process, the amount of buffer required to achieve a certain drop probability when the arrival rate of packets is more than 90% of the total capacity of the router, reduces by a factor of 4 if a shared memory is used.

In this section we establish that our schedulers make very effective use of memory. In Section 2.5.1 we show that the total memory used by our schedulers is very close to the minimum needed. In Section 2.5.2 we show that the number of memory banks used by our

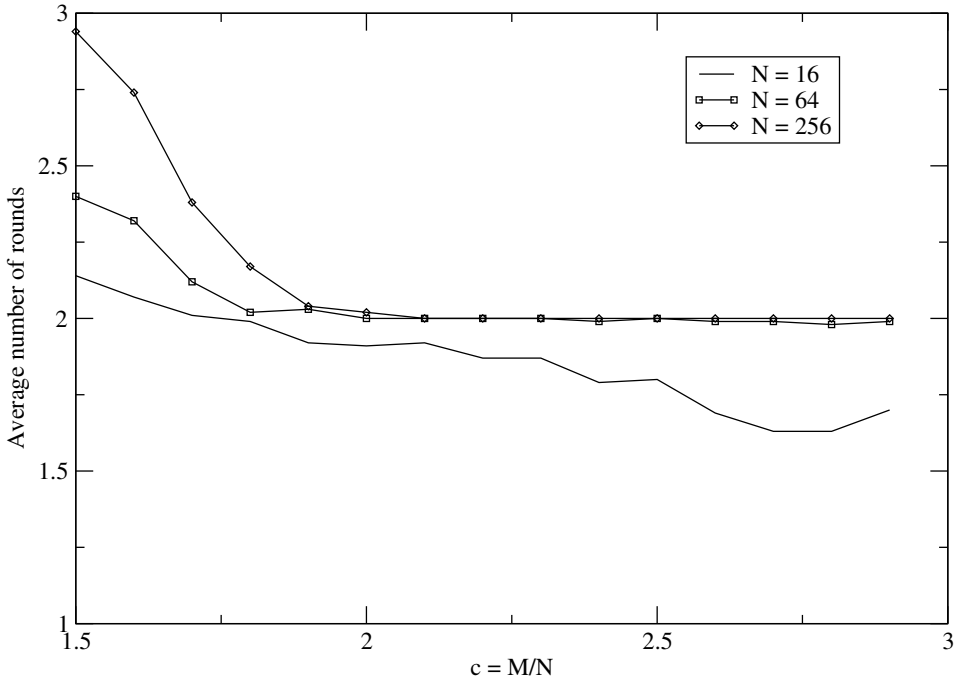


Figure 2.6: The average number of rounds needed to match all the inputs, plotted against ratio of number of memories to that of input ports (as observed in the simulations with **uniform Bernoulli traffic**, simulated for 100,000 cycles)

schedulers is also close to the best possible.

2.5.1 Load Balance

In SMS architecture, if a packet is matched to a memory that is full then it is dropped. One of the features of our algorithms is that they distribute packets evenly across the memory banks. This enables us to achieve the effect of a pure shared memory, i.e., when a packet is matched to a memory, there will be space in that memory to store the packet as long as the total number of packets stored in all the memories is not close to the cumulative size of all the memories. This is independent of any assumptions on the packet arrival process, as

shown in the theorem below.

Theorem 2.5.1 Consider an SMS switch with N input and output ports, M shared buffer memories, each of size K, and with each shared buffer supporting s writes per cycle. Let Q be given as an upper bound on the total number of packets in the memories in any cycle. If $K \geq Q/M + \sqrt{2csL \log M}$, with $c > 1$, then w.h.p. in M, our SMS scheduler can buffer packets for up to L cycles without dropping any packets.

Proof: The result follows through the use of Azuma's inequality [1] on the following martingale. Consider an arrival sequence of packets that leads to buffering of a total of R packets at the end of $T-1$ cycles and there is a packet at the input that has been matched to memory m to be stored there. At any time t, there could not be more than L packets stored in the system, so the time-stamp of a newly arrived packet can not be greater than $t + L$. Thus, at any given time, there can not be any packet in the buffers that arrived L cycles earlier. Let U_i be the map that maps packets arrived at $(T - L - 1 + i)$-th cycle to the memories that they are stored in. Let $\mathbf{U} = (U_1, U_2 \cdots U_L)$ and $V_m(\mathbf{U})$ be the random variable denoting number of packets stored in memory m at the end of $(T-1)$-th cycle. Note that since all the packets in memory arrived within last L cycles, $\mathbf{U}$ has sufficient information to compute V_m. Since there is no special bias for any of the memories, $\mathrm{E}(V_m \mid R) = R/M$. Define a sequence of random variables,

$$W_i = \mathrm{E}(V_m | U_1, U_2 \ldots U_i).$$

Since $\mathrm{E}(W_i \mid W_{i-1}) = W_{i-1}$, the sequence of random variables W_i is a martingale, and the result follows. Furthermore if $\mathbf{U}$ and $\mathbf{U}'$ differ in only one of the map U_i for $(T-L-1+i)$-th cycle, at most one packet could be stored in memory m in that cycle. Therefore V_m satisfies Lipschitz condition, i.e., $|V_m(\mathbf{U}) - V_m(\mathbf{U}')| \leq 1$. Thus, using Azuma's inequality, we get that

$$Pr\left[|W_L - W_0| > L^{1/2+\delta}\right] < e^{-L^{2\delta}/2}$$

We know $W_L = E(V_m \mid U_1, U_2 \ldots U_L) = V_m(\mathbf{U})$ and $W_0 = \mathrm{E}(V_m) = R/m$. Thus,

$$Pr\left[V_m \leq R/M + L^{1/2+\delta}\right] \geq 1 - e^{-L^{2\delta}/2}.$$

Now if an output queued switch does not drop packets then at least one of its N buffers must have space for the packets. Thus, the number of buffered packets, R, is no more than LN. Therefore with high probability number of packets in the memory m is less than $LN/M + L^{1/2+\delta}$. Now since each memory has space for $LN/M + L^{1/2+\delta}$ packets, the probability of drop of a packet due to buffer overflow is at most $e^{-L^{2\delta}/2}$. By union bound, the probability of drop of any of the packets is at most $Ne^{-L^{2\delta}/2}$. Thus, with high probability no packet will be dropped. ■

The following corollary follows from the theorem using $Q = LN$. Note that in general, an output queued switch will use a conservative value for L to allow for occasional bursts of traffic for a single output. Thus, the value of Q in the above theorem is typically much smaller than LN, and hence our scheduler would typically make much better use of the memory than a corresponding output-queued switch. Also, note that since typically $Q/M \gg M \gg \log M$, the value of K can be chosen to be only very slightly larger than Q/M, the minimum size needed, and the packet drop probability could be held very small even if L is made very large. Note also that the value of L in the above theorem is limited in only a weak way by the upper bound placed on the value on Q even if the value of K is to be held close to Q/M.

Corollary 2.5.2 Consider an SMS switch that emulates an output-queued switch with N ports and output buffer size L with M shared buffers, each of size K, and each supporting s shared writes per cycle. If $K \geq LN/M + \sqrt{2csL\log M}$, where $c > 1$ is a constant, then with high probability, both of our schedulers will not drop a packet that will not be dropped by that output-queued switch.

The total output buffer memory used by the output queued system is LN. By the above theorem, if the total amount of memory across the M memories in our SMS scheduler

is $LN + ML^{(1/2+\delta)}$, then no memory is assigned more than L packets w.h.p. Since $M = O(N)$ and typically $L \gg N$, the amount of memory used by our scheduler is within a $o(LN)$ additive term of the amount used by the output queued router. Further, an output queued scheduler with buffer size of L will start dropping packets when the average buffer size is well below L. In contrast, due to the shared memory and the randomized strategy used in our scheduler, our results are based only on the total number of packets in the memory, not on the maximum number of packets at any output queue.

Simulation studies of load balancing across memories

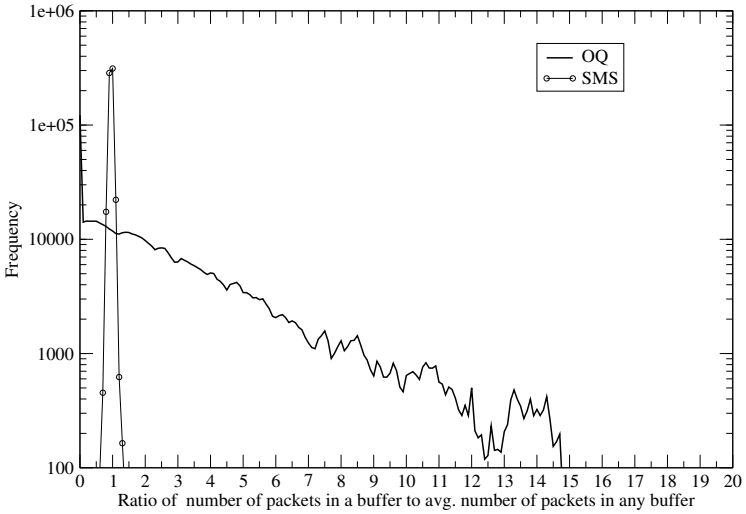
To study this theoretical prediction, we simulated both SMS and output queued routers with 64 input and output ports and bursty traffic (geometrically distributed bursts for randomly chosen outputs). For each cycle, we measured the ratio of number of packets in a buffer to that of average number of packets in each buffer. Ideally, if all buffers are equally full, all measurements should be close to one. A spread in these numbers indicates that buffers are not evenly occupied. Figure 2.7(a) shows distribution of this measurement for both SMS and OQ for bursty traffic. Here, the plot for SMS buffers is concentrated around one, indicating that packets are evenly balanced across all buffers, while for OQ, a buffer can have as much as 15 times the average number of packets. Figure 2.7(b) shows similar results for uniform Bernoulli arrivals. Since traffic arrives uniformly at all outputs, in this case the output queues also remain balanced and small, but even here, the distribution of packets across buffers is more balanced in the SMS router than in the OQ router.

2.5.2 Number of Memory Banks

Even though the cumulative size of memories in an SMS architecture can be close to that of an output-queued router, having a large number of small memories is slightly more expensive than having a small number of large memories.

We have shown that that $(1 + \lceil 1/s \rceil + \epsilon)N$ memories are sufficient for an SMS

(a) Bursty traffic



(b) Uniform Bernoulli traffic

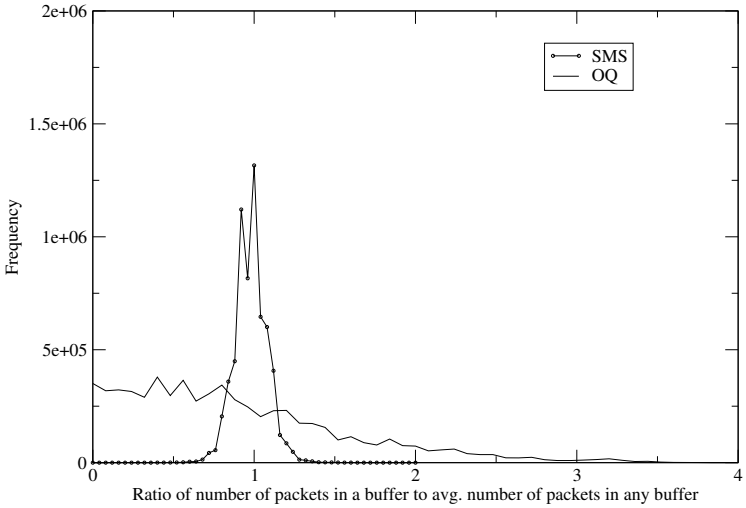


Figure 2.7: Distribution of the ratio of number of packets in a buffer to the average number of packets across all the buffers in that cycle (from simulations of 64 port SMS and OQ routers with (a) bursty traffic and (b) uniform Bernoulli traffic). Both systems were simulated for 100,000 cycles. A spread in the curve around one indicates that the packets are not evenly distributed across different memory banks. In both simulations we used $N = 64$ and $M = 128$. The average number of packets in the buffer for bursty arrivals was 4218 while in uniform arrivals it was 610.

router with speed-ratio s to emulate an output-queued router. It is natural to investigate how many memories are actually necessary. First, we examine what an off-line algorithm can achieve.

Lemma 2.5.3 If an algorithm has knowledge of the complete arrival sequence then N memories are sufficient to store the packets while satisfying arrival and departure conflicts.

Proof: Construct a bipartite multi-graph $G(V, W, E)$ in which the set of vertices V represent arrival times of packets, the set of vertices W represent the departure times of packets and one edge $(v, w) \in E$ is present for every packet that arrives at time v and departs at time w. Since at most N packets arrive at any cycle and at most N packets depart every cycle the maximum degree of any vertex in G is N. Thus, by Birkhoff's theorem [47, Page 40] it can be edge-colored using N colors and packets belonging to every color-class can be stored in one memory. ■

The requirement on N memories is also trivially a lower bound since there are potentially N new packets in a cycle.

Of course, in the context of a router, the algorithm has to operate on-line. Now we look at the absolute minimum number of memory banks that is required if an adversary is allowed to place packets in the memories.

Lemma 2.5.4 If an adversary places packets in the memory then it is necessary to have $N + \lceil (N-1)/s \rceil$ memories in order to satisfy arrival and departure constraints.

Proof: Consider the case where at every cycle $T < N^2 - 1$, exactly 2 packets arrive for output $(T \text{mod}(n-1)) + 1$, one packet arrives for every output o such that $o \neq (T \text{mod}(n-1)) + 1$ and $o < N$, and no packet arrives for output N. At the cycle $N^2 - 1$, the total number of arrivals at each output between 1 to $N-1$ would be $N^2 + N$, but the total number of packets that departed through each output would be $N^2 - 1$. Thus, there would be $N + 1$ packets in the memory for each output from 1 to $N - 1$. Hence, for each of the next $N + 1$ cycles, we will have $N - 1$ packets scheduled to depart. An adversary could

choose a set B of $N-1$ memories and place all these packets into the memories in B such that each memory stores one packet of each time-stamp between N^2 and N^2+N. Now if N packets arrive all destined for output N, then each packet will have a departure conflict with each memory in B. Thus, all the new packets must be stored in some memory that is not in B and no 2 packets can be stored in same memory. Therefore there must be additional N memories. Hence, we need $2N-1$ memories to store the packets. ■

Since our algorithm controls the placement of packets in the memory it is possible that a good algorithm can make do with a smaller number of memory banks than the bound in Lemma 2.5.4. We now show that it is impossible for an SMS router with $N+o(N)$ memories to behave identically to an output-queued router, regardless of how sophisticated its scheduling algorithm is.

Theorem 2.5.5 There is no deterministic algorithm that can match any sequence of packet arrivals to memories while satisfying arrival and departure constraints if the number of memories is $M=N+\Delta$ and $\Delta < N/8$. Furthermore, for any randomized algorithm there exists an arrival sequence for which it will fail with probability at least $1/2$.

In order to prove the theorem we will use a set of lemmas that show that if we have a sequence of subsets of size close to half of the original set such that any two consecutive sets are disjoint, then any pair sets with even sequence number have a significant intersection.

Lemma 2.5.6 If $X, Y, Z \subseteq [N+\Delta]$ such that $|X| = |Y| = N/2$ and $X \cap Y = Y \cap Z = \emptyset$ then $|X \cap Z| \geq N/2 - \Delta$.

Proof:

$$
\begin{aligned}
|Z| &= |Z \cap X| + |Z \cap X^c| \\
|Z \cap X| &= |Z| - |Z \cap X^c| = N/2 - |Z \cap X^c| \\
Z \cap Y^c &= Z && ; Z \cap Y = \emptyset \\
Z \cap X^c &= Z \cap Y^c \cap A^c \\
|Z \cap X^c| &\leq |Y^c \cap X^c| = |(Y \cup X)^c| \\
&= |(Y \cup X)^c| = N + \Delta - |X| - |Y| = \Delta \\
|Z \cap X| &\geq N/2 - \Delta
\end{aligned}
$$

■

Lemma 2.5.7 For any three sets X,Y,Z of size $N/2$ if $|X \cap Y| \geq N/2 - \alpha$ and $|Y \cap Z| \geq N/2 - \beta$, then $X \cap Z \geq N/2 - \alpha - \beta$.

Proof: By splitting Y into $Y \cap X$ and $Y - (X \cap Y)$ we get,

$$|Y \cap Z| = |(Y \cap X) \cap Z| + |(Y - (X \cap Y)) \cap Z|$$

Thus,

$$
\begin{aligned}
|(Y \cap X) \cap Z| &= |Y \cap Z| - |(Y - (X \cap Y)) \cap Z| \\
&\geq |Y \cap Z| - |(Y - (X \cap Y))| \\
&= |Y \cap Z| - (|Y| - |X \cap Y|) \\
&= N/2 - \beta - \alpha
\end{aligned}
$$

Therefore, $|X \cap Z| \geq N/2 - \beta - \alpha$ ■

Lemma 2.5.8 For any series of sets $S_0, S_1, \ldots, S_{2m} \in [N + \Delta]$ if $|S_i| = N/2$ and $S_i \cap S_{i+1} = \emptyset$, then $|S_0 \cap S_{2m}| \geq N/2 - m\Delta$.

Proof: The base case when $m = 1$ follows from Lemma 2.5.6. Let the lemma be true for some $m = p$. Thus, $|S_1 \cap S_{2p}| \geq N/2 - p\Delta$ and $|S_{2p} \cap S_{2p+2}| \geq N/2 - \Delta$. Therefore from Lemma 2.5.7, we get $|S_1 \cap S_{2(p+1)}| = N/2 - (p+1)\Delta$. ■

We can now prove Theorem 2.5.5. We will do so by defining two packet arrival sequences such that based on choices made by any algorithm, an adversary can always choose one of the arrival sequence for algorithm to fail if $\Delta < N/8$.

Assume the number of outputs is even. Let O_1 be a set of $N/2$ outputs and O_2 be the remaining set of outputs. Define a_i (b_i) to be the set of packets that depart at time i and are destined for an output in O_1 (O_2). Our arrival process is such that $|a_i| = N/2$ or 0 and all the packets for any set a_i arrive in the same cycle. Similarly, $|b_i| = N/2$ or 0 and all the packets in any set b_i arrive in the same cycle.

Now we will present two arrival sequences. The two arrival sequences are described in Table 2.3. Both sequences have a common prologue till time 9 as described in the first column of the table. The second and third columns describe the packets that arrive in sequence 1 and sequence 2 respectively after prologue. A dash in the input column indicates that no packets arrived at those $N/2$ inputs.

It is easy to verify that the time-stamp assignments are consistent with output queuing. We will use A_i^* (B_i^*) to represent the set of memories that the packet of a_i (b_i) will be stored in, where the superscript $*$ is either p, 1 or 2 based on whether the set of packets correspond to prologue, sequence 1 or, sequence 2 respectively. Since all the packets departing together must be stored in different memories, if $a_i \neq \emptyset$ then $|A_i^*| = |a_i| = N/2$. Similarly, if $b_i \neq \emptyset$ then $|B_i^*| = N/2$.

For notational convenience, we introduce the infix binary relational operator $\not\leftrightarrow$ denoting set disjointness, i.e., $U \not\leftrightarrow V$ iff $U \cap V = \emptyset$. From arrival time constraints we get $A_{11}^p \not\leftrightarrow A_{12}^p$, $B_{12}^1 \not\leftrightarrow B_{13}^1$, $B_{12}^2 \not\leftrightarrow B_{13}^2$, and $A_{14}^2 \not\leftrightarrow B_{11}^2$, and from departure time constraints we get $A_{11}^p \not\leftrightarrow B_{11}^2$, $A_{12}^p \not\leftrightarrow B_{12}^1$, $A_{13}^p \not\leftrightarrow B_{13}^1$, $A_{13}^p \not\leftrightarrow B_{13}^2$, and $A_{14}^2 \not\leftrightarrow B_{14}^2$.

Now since there are a total of $N+\Delta$ memories and A_{11}^p is connected to B_{11}^2 through a chain of 8 $\not\leftrightarrow$ relations, from Lemma 2.5.8 we set $B_{11}^2 \cap A_{11}^p \geq N/2 - 4\Delta$. But we know that $B_{11}^2 \cap A_{11}^p = \emptyset$. Thus, $N/2 - 4\Delta \leq 0$ or $\Delta \geq N/8$.

Therefore we conclude that if $\Delta < N/8$ any deterministic algorithm will fail.

Furthermore, if any randomized algorithm, chooses A_{11}^p and A_{13}^p such that it works correctly for sequence 1 with probability θ, then it must fail for sequence 2 with probability θ. Thus, the worst case probability of failure for any randomized algorithm is at least $\max(\theta, 1-\theta) \geq 0.5$.

The key to this proof are the two arrival sequences. We would like to thank Felice Balarin for suggesting the sequences.

Prologue			Sequence 1			Sequence 2		
time	input		time	input		time	input	
1	a_1	a_2	10	b_{11}	$-$	10	b_{11}	a_{14}
2	a_3	a_4	11	b_{12}	b_{13}	11	b_{12}	$-$
3	a_5	a_6				12	b_{13}	b_{14}
4	a_7	a_8						
5	a_9	a_{10}						
6	a_{11}	a_{12}						
7	b_7	b_8						
8	b_9	b_{10}						
9	a_{13}	$-$						

Table 2.3: Adversarial arrival sequence

Chapter 3

Pipelined Scheduling

If the switching fabric is sufficiently fast, the throughput of a switch critically depends upon how long it takes to do the scheduling every cycle. Thus if we can reduce the number of rounds required by RiPSS, we can potentially increase the throughput of a switch.

When multiple rounds of the BMP are used, the memories and inputs that are matched in earlier rounds remain idle till the end of the cycle. In this Chapter we study a series of algorithms that use pipelining to make better use of the interconnect, thereby executing only a constant number of rounds of communication per cycle.

The main intuition behind the pipelined scheme is that is that when a memory is matched to an input, instead of sitting idle it can work towards matching future packets. Similarly once an input port has matched a packet it can work towards matching future packets. Since we do not know about the future, we put a small buffer at the input that buffers packets for a small number of cycles such that the scheduler always has future packets to match.

The pipelined schedulers use multiple cycles to construct a matching for each set of packets that arrive together. However, matchings are constructed for multiple sets of packets simultaneously in a pipelined fashion. Consequently, the amount of computation per cycle reduces but packets wait for D cycles at the inputs before they are transferred to

the memories. The value D is the *latency* of the pipelined scheduler. The input buffer size I equals D, and packets are stored FIFO.

The goal of designing a pipelined scheduler is to minimize (a) amount of work done per cycle, (b) the number of cycles packets have to wait at the input (D), and (c) the hardware cost. In a naïve implementation of pipelining we can just have D parallel hardware units, such that the i-th unit is performing the i-th round of BMP for packets that arrived i cycles earlier. However, that would increase the hardware requirement excessively. Here we propose a series of pipelined scheduling algorithms. We will generically refer to these scheduling algorithms as PRiPSS (Pipelined RiPSS).

The departure time assignment for all the algorithms is identical. Let P_1^o through $P_{c^o}^o$ be the packets destined for output port o that arrived during cycle T and let them be ordered according to the id of the input port they arrived. We maintain an array $earliest[1 \cdots N]$ to keep track of earliest time-stamp available for any output, after taking latency into account. The time-stamp of packet P_i^o is then set to $earliest[o] + i + D$ and $earliest[o]$ is updated to $\max(earliest[o] + c^o, T)$.

In cycle T, the packets that arrived between cycles $T - D$ and T are in the input buffers and at the end of cycle T, the packets that arrived at cycle $T - D$ that are matched are transferred to the memories.

We use the concept of active packets to reduce the amount of work done at the inputs. In a given cycle, each input port has an initial sequence of packets in its buffer that have been matched to some memory by the scheduling algorithm in earlier cycles, and the remaining packets are not yet matched by the scheduling algorithm. At any point in the scheduling algorithm, the first unmatched packet in each buffer is called the *active* packet for the step, and the scheduling algorithm tries to match that packet to one of the compatible memories.

Now we will present a series of algorithms that lead up to our final pipelined algorithm, PRiPSS-v3.The first algorithm we present, requires $\log \log N$ stages and a constant

number of rounds per stage. In the second algorithm we will eliminate some of the rounds in each cycle. In PRiPSS-v3 the number of pipeline stages reduces to $\log^* N$.

3.1 PRiPSS-v1

One way to implement pipelining would be that each memory tries to match one active packet for each arrival time, in parallel. This would require more control hardware at the memories. We eliminate this requirement by slightly modifying the algorithm. In PRiPSS-v1, each memory sends just one request message to one of the inputs with a compatible active packet. However, we pay a price in terms of latency. Now the number of pipeline stages needed for scheduling grows as $O(\log \log N)$. Below we formally describe PRiPSS-v1.

A *stage* of the pipeline executes the three steps in the following pipelined matching procedure ω times, where ω is an integer constant to be defined later in the analysis.

PRiPSS-v1: Procedure for Cycle T

(a) Execute the following three phases:

Phase 1:

(i) In parallel each new active input broadcasts to all memory banks the time-stamp of its active packet together with its arrival time mod D.

(ii) In parallel each memory sends a message to a random compatible input port.

(ii) In parallel each input port picks a memory bank that sent it a message and assigns its active packet to that memory bank. It then replaces its matched active packet by the first unmatched packet in its buffer.

Phase 2: Repeat phase 1 on the current active packets.

Phase 3: Repeat phase 1 *only on current active packets that arrived in cycle* $T - D$.

(b) Transfer all matched packets that arrived in cycle $T - D$ to the memory banks.

The above algorithm, PRiPSS-v1, as well as the ones presented in following sections, PRiPSS-v2 and PRiPSS-v3, match all the packets with high probability. The proof of this fact for all the algorithms are similar in spirit. We will only give the proof for PRiPSS-v3. Here we will just state the following result.

Theorem 3.1.1 At the end of any cycle, PRiPSS-v1 will match all the packets that arrived D cycles earlier, w.h.p. in N, if for a suitable constant c, $D = c \cdot \log \log N$.

3.2 PRiPSS-v2

In PRiPSS-v1, we introduced an additional phase 3 where memories concentrated on packets that arrived D cycles earlier in order to guarantee that everything gets matched in the last step. In PRiPSS-v2, we will eliminate this step by increasing the probability of a memory sending request to an active packet that arrived D cycles earlier.
In the following, δ is a suitably small constant.

(a) **Phase 1:**

(i) In parallel each new active input broadcasts to all memory banks the time-stamp of its active packet together with its arrival time mod D.

(ii) In parallel each memory bank m sends a message to an input port chosen as follows: Let A_m be the set of inputs compatible with m that have an active packet that arrived in cycle $T - D$. If A_m is non-empty then with probability δ one of the inputs in A_m is chosen and with probability $1 - \delta$ some other compatible input is chosen; otherwise the input is chosen uniformly at random from all compatible inputs.

(ii) In parallel each input port picks a memory bank that sent it a message and assigns its active packet to that memory bank. It then replaces its matched active packet by the first unmatched packet in its buffer.

Phase 2: Repeat phase 1 on the current active packets.

(b) Transfer all matched packets that arrived in cycle $T - D$ to the memory banks.

Theorem 3.2.1 At the end of any cycle, PRiPSS-v2 will match all the packets that arrived D cycles earlier, w.h.p. in N, if for a suitable constant c, $D = c \cdot \log \log N$.

3.3 PRiPSS-v3

Using the ideas in PRiPSS-v2, we further skew the probabilities of memories making request to different inputs. In PRiPSS-v3, we reduce the value of D to $O(\log^* N)$. Below, we formally describe the algorithm.

A *stage* of the pipeline executes the three steps in the following pipelined matching procedure ω times, where ω is an integer constant to be defined later in the analysis.

Pipelined Matching Procedure

(a) The input ports perform a transmit step in which each input port broadcasts to all the memories the time-stamp of its active packet (as in RiPSS) together with its arrival time mod D.

(b) In parallel, each memory bank picks an index i between 0 and D, and matches itself to a random compatible input with exactly i unmatched inputs. The index i is chosen with probability p_i, where $p_i = 1/2^{i+1}$ if $i < D$ and $p_D = 1/2^D$.

(c) Each matched active packet is replaced by the first unmatched packet in its buffer.

Finally, all matched packets that arrived in cycle $T-D$ are transferred to the memory banks, and this concludes the stage. Any unmatched packet that arrived in cycle $T - D$ is dropped.

3.3.1 Analysis

Our analysis assumes that $M = (1 + (1/s) + \epsilon)N$, where ϵ is an arbitrarily small positive constant. The complete analysis is in the Appendix. Here, we present a simplified analysis

for the case when ϵ and s are both 1. Let $z_r = \frac{N}{2\uparrow\uparrow r}$. With $s = 1$ and $\epsilon = 1$, the number of unmatched inputs goes down by a factor of 2 w.h.p. in each iteration of the BMP, and after r iterations of the BMP in the non-pipelined setting, the number of unmatched packets is $\leq z_r$ if $z_r > \sqrt{N}$. Let $D = \log^* N$. Let $\omega = 2$, i.e., a stage of the pipelined scheduler consists of two iterations of the pipelined matching procedure.

Let $Q_i(T)$ be the set of input ports that have i unmatched packets at the start of cycle T, and let $q_i(T) = |Q_i(T)|$. Let $s_i(T) = \sum_{k=i}^{D} q_k(T)$. We define a predicate $\Lambda_0(T)$ to be true iff for all $i \leq D$, $s_i(T) \leq z_i$.

Theorem 3.3.1 If $\Lambda_0(T-1)$ is true then w.h.p. in N, $\Lambda_0(T)$ is true.

Proof: Consider the start of cycle T. Note that for any input port with i unmatched packets, the number of packets that can be matched at that port during cycle $T-1$ is 0, 1, or 2 (since we have assumed that $\omega = 2$). Let $r_i(T-1)$ be the number of inputs that had i or $i-1$ unmatched packets at the start of cycle $T-1$ and have at least $i-1$ unmatched packets at the end of cycle $T-1$. Since one new packet arrives at each input port at the start of cycle T, we have

$$s_i(T) = \sum_{k=i}^{D} q_k(T) \leq \sum_{k=i+1}^{D} q_k(T-1) + r_i(T-1) \leq s_{i+1}(T-1) + 3z_{i+1}$$

The last equation above uses the inequality $r_i(T-1) \leq 3z_{i+1}$. We can establish this as follows:

Let n_1 be the number of inputs in $Q_i(T-1)$ that are unmatched after the first iteration of stage $T-1$, let X be the set of inputs that have $i-1$ unmatched packets after the first iteration of stage $T-1$, and let n_2 be the number of inputs in X that are unmatched after the second iteration of stage $T-1$. Then $r_i(T-1) = n_1 + n_2$.

Since $q_i(T-1) \leq s_i(T-1) \leq z_i$ (by the induction assumption), we have $n_1 \leq z_{i+1}$ (since the number that did not receive a match in an iteration of stage $T-1$ is the same as that derived for RiPSS since the pipelined algorithm executes the BMP in parallel for each i.)

For n_2 we note that $|X| = x_1 + x_2$, where x_1 is the number of inputs that had i unmatched packets at the start of cycle $T-1$, and have $i-1$ unmatched packets after the first iteration, and x_2 is the number of inputs that had $i-1$ unmatched packets at the start of cycle $T-1$ and continue to have $i-1$ unmatched packets after the first iteration. Clearly, $x_1 \leq q_i(T-1)$, and $x_2 \leq z_i$ by the behavior of the basic matching process on inputs that had $i-1$ matched packets at the start of cycle $T-1$. Hence, $|X| \leq q_i(T-1)+z_i \leq z_i+z_i \leq 2z_i$. Hence, $n_2 \leq 2z_{i+1}$. Hence, $r_i \leq 3z_{i+1}$.

So we have

$$s_i(T) \leq s_{i+1}(T-1) + 3z_{i+1} \leq 4z_{i+1} < z_i$$

■

Corollary 3.3.2 W.h.p. in N, all packets that arrived in cycle $T-D$ have been matched by end of cycle T.

Proof: From the theorem, $q_D(T-1) = s_D(T-1) \leq \max(p_D, \sqrt{N}) = \sqrt{N}$. During the first iteration of cycle T, the basic matching procedure is applied to these $\sqrt{N}$ inputs. Hence, w.h.p. in N, all packets that arrived in cycle $T-D$ are matched after this step, and certainly by the end of cycle T. ■

Since $\Lambda_0(0)$ is trivially true, by Theorem 3.3.1, we can argue inductively that $\lambda_0(T)$ is true when $T = O(N)$. However, as T grows large, the probability that $\lambda_0(T)$ will continue to be true becomes small and then we can no longer guarantee that all the packets that arrived in cycle $T-D$ will be matched at the end of cycle T. However, our algorithm has a "self-stabilizing" property, i.e., if $\Lambda_0(T)$ becomes false for some T, within $O(\log N)$ cycles the input queues get back to a state where the predicate Λ_0 is true.

We define a series of predicates $\Lambda_j(T)$ such that $\Lambda_j(T)$ is true iff for all i, $s_i(T) \leq (\phi)^j p_i$ for some constant $\phi > 1$. Note that $\Lambda_j(T)$ implies $\Lambda_k(T)$ if $k \geq j$.

Theorem 3.3.3 If $j > 0$ and $\Lambda_j(T-1)$ is true then, w.h.p, $\Lambda_{j-1}(T)$.

Proof:(sketch) Recall that in the proof of Theorem 3.3.1, we proved that $s_i(T) \leq 3z_i$. Using a similar argument here, we can prove that if $\Lambda_j(T-1)$ is true, then $s_i(T) \leq 3\phi^j z_i$. Now for $c > 3\phi$ we get $s_i(T) \leq c\phi^{j-1} z_i$. If $j > 0$ then $s_0 \leq p_0$ trivially. Hence, $\Lambda_{j-1}(T)$.

■

Now since $\Lambda_{\log_\phi N}(T)$ is always true, in $\log_\phi N$ steps we get back to a state where $\Lambda_0(T)$ is true. This establishes the self-stabilizing feature of our pipelined algorithm.

3.4 Simulation of PRiPSS-v3

We ran simulations of PRiPSS-v3 for 100,000 cycles with Bernoulli arrival sequences, for number of ports N varying between 16 and 4096, and number of memories M varying from $1.6 \cdot N$ to $2.5 \cdot N$.

PRiPSS-v3 ran without any packet being dropped in any of the runs even when the number of rounds in each cycle was held to $\omega = 2$. For $N \leq 2048$, we needed $D = 2$. For $N = 4096$, we needed $D = 3$.

We simulated PRiPSS-v1, PRiPSS-v2, and PRiPSS-v3 for 100,000 cycles with uniform Bernoulli arrivals and varied D. For PRiPSS-v2, we found $\delta = 0.5$ gave the best results. Table 3.1 shows the minimum value of D needed such that all the packets are matched in the simulations. Note that the algorithm for PRiPSS-v1 requires 3 rounds of communication between memories and inputs while PRiPSS-v2 and PRiPSS-v3 use only 2 rounds in our simulations.

It appears that PRiPSS-v3 is an attractive alternative to the basic non-pipelined RiPSS and performs better than PRiPSS-v1 and PRiPSS-v2. It placed every packet in memory using just 2 rounds per cycle while keeping the latency to only two cycles for $N \leq 2048$, and using only $M = 1.6N$ memory banks. While PRiPSS-v2 also requires only 2 rounds per cycle, it needs 3 stages for $N \in [1024, 2048]$ and $1.6N \leq M \leq 2N$ in contrast to PRiPSS-v3 that requires only 2 stages.

$M/N =$	1.6	2.0	2.5	1.6	2.0	2.5	1.6	2.0	2.5
N	PRiPSS-v3			PRiPSS-v1			PRiPSS-v2		
16	2	2	2	2	2	2	2	2	2
32	2	2	2	2	2	2	2	2	2
64	2	2	2	2	2	2	2	2	2
128	2	2	2	2	2	2	2	2	2
256	2	2	2	2	2	2	2	2	2
512	2	2	2	2	2	2	2	2	2
1024	2	2	2	3	3	2	3	3	2
2048	2	2	2	3	3	2	3	3	2
4096	3	3	2	3	3	2	3	3	2

Table 3.1: Number of stages needed for different values of M and N in PRiPSS-v3, PRiPSS-v1, and PRiPSS-v2, $\delta = 0.5$. These values were obtained by simulation for 100,000 cycles with uniform Bernoulli traffic.

Chapter 4

Scheduling a combined input-output switch

For latency insensitive applications, the prime concern is to achieve stability at minimum cost. IQ (*c.f.* Section 1.4.1) architecture is one of the simplest architectures that can achieve stability, however, the corresponding scheduling algorithm is too complex. Thus, we examine the possibility of using little speed-ratio such that scheduling problem becomes less complex. As a result of using speed-ratio, we need buffering at output side as well. First we present an extremely simple scheduling algorithm called Randomized Matching (RM). We show that RM cannot be stable with any amount of constant (independent of N) speed-ratio.

4.1 Architecture

The packets are buffered at both input ports as well as output ports. Each input maintain N logical queues each corresponding to packets destined for a particular output port; these queues are commonly known as Virtual Output Queues (VOQ) [28]. Let Q_{ij} denote the VOQ at input i corresponding to output j. As the packets arrive they are stored in one of these VOQs. The scheduler constructs a matching of inputs to outputs and then programs

the crossbar to make transfers. At each input i that is matched to some output j the packet at the head of VOQ Q_{ij} is read and transmitted to output j through the crossbar. If the speed-ratio is s, each cycle s matches are constructed and corresponding packets are transfered to the output.

As in every cycle only one packet can be transmitted and potentially s packets can arrive at the output, each output also maintains a queue of packets to send. At the end of every cycle each output with a nonempty queue transmits the packet at the head of its queue. For this reason, it is more accurate to describe the system as combined input-output queued switch as opposed to IQ.

Even a trivial matching algorithm that just chooses one of the nonempty VOQs and matches the corresponding inputs and outputs would assure stability with speed-ratio of N. Our goal is to come up with a smart algorithm with low running time that can assure stability with little (constant) speed-ratio. We first construct a very simple randomized algorithm called randomized matching. This is essentially the BMP used in the context of SMS (*c.f.* Section 2.4).

4.2 Randomized Matching

This algorithm works in two phases.

1. **Request**: Each input i that has at least one nonempty queue selects one of the outputs for which it has a packet, uniformly at random, and sends a request message to that output.

2. **Grant**: Each output that received one or more requests in the previous phase selects one of the requesting inputs, uniformly at random to send a grant message.

The input-output pairs that exchange a request and a grant message are considered matched. Assuming that probability of large number of inputs sending a request to same output would be small, at least a small fraction of outputs should get a request and hence

get matched to one of the inputs. Let that fraction be f. Thus, if we provide a speed-ratio of $\lceil 1/f \rceil$ then we should be able to achieve stability. However, we discovered that certain admissible arrival patterns exist for which f can be extremely small. In [36] we have shown that there exists an adversarial admissible traffic pattern for which at least a speed-ratio of $\sqrt{\log N}$ is necessary for stability.

The main reason why the algorithm performs so badly is that we construct the adversarial traffic in a way that there are several input-output pairs for which packets arrive at a slow rate while there are a small number of input-output pairs for which packets arrive at a much faster rate. Since there is always a large number of input-output pairs competing with these pairs with large rate, they will seldom get a chance to transfer a packet resulting in instability.

The above argument suggests that the matching algorithm should be biased towards the input-output pairs for which packets arrive at faster rate than others. A good indicator of the rate is the queue length. This intuition led to development of the following algorithm.

4.3 Weighted Random Matching (WRM)

Let $d_{ij}(t)$ be the number of packets in the queue Q_{ij} at time t. Let $D(t) = (d_{11}, d_{12} \ldots d_{NN})$ be the vector representing state of the system. The number of packets queued at the input i is defined to be $l_i(D) = \sum_j d_{ij}$ and the number of packets queued for output j across all inputs is defined to be $r_j(D) = \sum_i d_{ij}$. Define $w_{ij}(D) = d_{ij}(D)/l_i(D)$ to be the *weight* of queue Q_{ij}. If $l_i(D)$ is zero, then all weights $w_{ij}(D)$ are defined to be zero.

WRM also proceeds in two phases.

1. **Request**: Each input i that has at least one nonempty queue randomly selects one of the outputs such that the probability of choosing output j is equal to $w_{ij}(D)$. A request message is sent to the selected output.

2. **Grant**: Each output that received one or more requests in the previous phase selects

one of the requesting inputs, uniformly at random to send a grant message.

Input-output pairs that exchange a request and grant message are considered matched and the packet at the head of corresponding VOQ is transfered through the crossbar.

A similar algorithm was also studied by Leonardi *et al.* in [26]. However, they do not consider variable length packets and they do not give any bound on expected latency.

4.3.1 Proof of stability of WRM

We will prove that WRM achieves 100% throughput. We will use Foster's criterion [6] to prove this. Specifically, Foster's theorem says that if there exists a potential function and a bounded compact set outside of which the potential function is always expected to decrease, then the system will never diverge.

Theorem 4.3.1 [Foster's criterion [6]] Let $M(k)$ be the state vector of an aperiodic irreducible discrete time Markov chain with a countable state space E. Let $\Phi : E \rightarrow \mathcal{R}^+$ be a function defined over the state space mapping it to the set of positive real numbers and let $V = \{s \in E | \Phi(s) < C\}$, where C is a positive constant. Suppose $\mathrm{E}[\Phi(M(k+1))|M(k)]$ exists and there exists a $\delta > 0$ such that if $M(k) \notin V$ implies

$$\mathrm{E}[\Phi(M(k+1) - M(k) \mid M(k)] \leq -\delta$$

then $M(k)$ will return to the set V infinitely often and $\lim_{k \rightarrow \inf} \mathrm{E}[\Phi(M(k))]$ is finite.

The key to the analysis is the following integer-valued function defined over state space of VOQs,

$$\Phi(D) = \sum_i l_i(D)^2 + \sum_j r_j(D)^2.$$

We will use Φ as a "potential function" for the system, specifically we show that if traffic is admissible, then $\Phi(D(t))$ can never grow to infinity. Since $\Phi(D(t))$ is bounded, each $l_i(D(t))$ must also be bounded. Hence, the queues never grow to infinity. We will prove boundedness of Φ using a sequence of very technical lemmas, and the intuition will be

given latter. In order to state the following lemma, first we observe that by regrouping sums $\sum_i l_i(D) = \sum_i \sum_j d_{ij}(D) = \sum_j \sum_i d_{ij}(D) = \sum_j r_i(D)$.

Lemma 4.3.2 If $\Phi(D) > 2Z^2$, then $\sum_i l_i(D) = \sum_j r_j(D) > Z$.

Proof: If $\Phi(D) > 2Z^2$, then either $\sum_i l_i(D)^2 > Z^2$ or $\sum_j r_j(D)^2 > Z^2$. In the former case since $l_i(D) \geq 0$, we argue that $(\sum_i l_i(D))^2 \geq \sum_i l_i(D)^2 > Z^2$. Hence, $\sum_j r_j(D) = \sum_i l_i(D) > Z$. The latter case is symmetric. ■

The following lemma deals with the decrease in potential function after one execution of WRM.

Lemma 4.3.3 If $D = (d_{ij})$ represents the state of queues of an $n \times n$ switch at the beginning of an execution of WRM and $D' = (d'_{ij})$ represents the state of the queues at the end of execution then $\mathrm{E}[\Phi(D) - \Phi(D') \mid D] \geq 2\sum_i r_j(D) - 2n$.

Proof: For notational convenience we will drop some syntactical elements when it is obvious from the context, e.g., we will use l_i instead of $l_i(D)$ and when we refer to D at time (a, b) we write $D(a, b)$ instead of $D((a, b))$.

Let U be the set of inputs that are busy and let V be the set of outputs that are busy during an execution of WRM. The probability that a non-busy input i sends a request message to output j is w_{ij}. Let o_{ij} be the random variable representing the number of inputs other than i that send a request to output j. Given that input i sends a request to output j and j is not busy, the probability of input i getting a grant from output j is $\frac{1}{1+o_{ij}}$. Thus, summing the probability of all disjoint events $(o_{ij} = x)$, the probability that input i gets matched to output j is

$$
\begin{aligned}
& w_{ij} \textstyle\sum_x \Pr[o_{ij} = x \mid D] \left(\frac{1}{1+x}\right) \\
= \quad & w_{ij} \mathrm{E}[\tfrac{1}{1+o_{ij}} \mid D] \\
\geq \quad & w_{ij} \tfrac{1}{1+\mathrm{E}[o_{ij} \mid D]} \quad ; \quad \text{Jensen's inequality [11]} \\
= \quad & \tfrac{w_{ij}}{1+\sum_{k \neq i, k \notin U} w_{kj}} \\
\geq \quad & \tfrac{w_{ij}}{1+\sum_{k \notin U} w_{kj}}
\end{aligned}
$$

Let $W_j = \sum_{i \notin U} w_{ij}$. Let s_{ij} be one if input i gets matched to output j and zero otherwise. Thus, if $i \notin U$ and $j \notin V$

$$\mathrm{E}[s_{ij}] \quad \geq \quad \frac{w_{ij}}{1 + W_j} \tag{4.1}$$

Furthermore, since busy inputs and outputs are always matched, if $i \in U$, then $\sum_j s_{ij} = 1$ and if $j \in V$, then $\sum_i s_{ij} = 1$.

Let Δ^- be the decrease in the potential function due to removal of the matched cells from VOQ. Thus,

$$\begin{aligned} \Delta^- &= \sum_i l_i^2 + \sum_j r_j^2 - \sum_i (l_i - \sum_j s_{ij})^2 + \sum_j (r_j - \sum_i s_{ij})^2 \\ &\geq 2\sum_i (l_i \cdot \sum_j s_{ij}) + 2\sum_j (r_j \cdot \sum_i s_{ij}) - 2n \end{aligned}$$

Thus, the expected decrease in the potential function

$$\begin{aligned} \mathrm{E}[\Delta^- \mid D] \quad \geq \quad & \mathrm{E}[2\textstyle\sum_{i \notin U}(l_i \sum_{j \notin V} s_{ij}) + 2\sum_{j \notin V}(r_j \sum_{i \notin U} s_{ij}) \\ & +2\textstyle\sum_{i \in U} l_i + 2\sum_{j \in V} r_j - 2n \mid D] \end{aligned}$$

Applying linearity of expectation, Equation 4.1, and rearranging order of summations we get the following:

$$\begin{aligned} &= 2\textstyle\sum_{i \notin U}(l_i \sum_{j \notin V} \mathrm{E}[s_{ij}]) + 2\sum_{j \notin V}(r_j \sum_{i \notin U} \mathrm{E}[s_{ij}])) - 2n \quad ; \text{ linearity of expectation} \\ &\geq 2(\textstyle\sum_{i \notin U}(l_i \sum_{j \notin V} \frac{w_{ij}}{1+W_j}) + \sum_{j \notin V}(r_j \sum_{i \notin U} \frac{w_{ij}}{1+W_j}) - n) \quad ; \text{ by (4.1)} \\ &= 2((\textstyle\sum_{i \notin U} \sum_{j \notin V} \frac{l_i w_{ij}}{1+W_j}) + (\sum_{j \notin V}(r_j \frac{W_j}{1+W_j})) - n) \\ &= 2(\textstyle\sum_{j \notin V} \sum_{i \notin U} \frac{d_{ij}}{1+W_j}) + (\sum_{j \notin V}(r_j \frac{W_j}{1+W_j})) - n) \\ &= 2(\textstyle\sum_{j \notin V} \frac{\sum_{i \notin U} d_{ij}}{1+W_j}) + (\sum_{j \notin V}(\frac{r_j W_j}{1+W_j})) - n) \\ &= 2(\textstyle\sum_{j \notin V} \frac{r_j}{1+W_j}) + (\sum_{j \notin V}(r_j \frac{W_j}{1+W_j})) - n) \\ &= 2\textstyle\sum_{j \notin V} r_j - 2n \end{aligned}$$

■

From the above lemma it follows that $\mathrm{E}[\Phi(D(c,1)) - \Phi(D(c,2)) \mid D(c,1)] \geq 2\sum_j r_j(D(c,1)) - 2n$. Now, with each execution of WRM, cells destined for any output can decrease at most by one. Hence, $r_j(D(c,2)) \geq r_j(D(c,1)) - 1$. Therefore $\mathrm{E}[\Phi(D(c,2)) - \Phi(D(c,3)) \mid D(c,1)] \geq 2\sum_j r_j(D(c,2) - 2n \geq 2\sum_j r_j(D(c,1) - 4n$. Thus, $\mathrm{E}(\Phi(D(c,1)) - \Phi(D(c,3))) \geq 4\sum_j r_j(D(c,1) - 6n$.

Now we will consider the increase in the potential function due to arrival of cells. Let $a_{ij}(c)$ be the number of cells that arrive at the input i for output j in the c-th cycle. Recall that $I_i(c) = \sum_j a_{ij}(c)$ and $O_j(c) = \sum_i a_{ij}(c)$. So the increase in the potential function during the 3rd step of the c-th cycle would be

$$\begin{aligned} & \textstyle\sum_i (l_i(c,3) + I_i(c))^2 + \sum_j (r_j(c,3) + O_j(c))^2 - \Phi(D(c,3)) \\ = \; & \textstyle\sum_i (2l_i(c,3)I_i(c) + I_i(c)^2) + \sum_j (2r_j(c,3)O_j(c) + O_j(c)^2) \\ \leq \; & \textstyle\sum_i (2l_i(c,1)I_i(c) + I_i(c)^2) + \sum_j (2r_j(c,1)O_j(c) + O_j(c)^2) \\ & \qquad \text{since } l_i(c,3) \leq l_i(c,1) \text{ and } r_j(c,3) \leq r_j(c,1) \end{aligned}$$

Now we are ready to prove our main theorems about stability under admissible traffic.

Theorem 4.3.4 WRM scheduler achieves 100% throughput for admissible traffic.

Proof: Consider an interval of T cycles, between $(c,1)$ to $(c+T,1)$. During this interval at most T cells could be transfered from any input or to any output. Similarly, no more than $T(1-\epsilon)$ cells are expected to arrive at any input or for any output. Thus, if $(c,1) \leq t < (c+T,1)$, then $l_i(c,1) - T \leq l_i(t) \leq l_i(c,1) + T$ and $r_j(c,1) - T \leq r_j(t) \leq r_j(c,1) + T$. So for any $u \in [c, c+T]$, $\Phi(D(u+1,1)) - \Phi(D(u,3)) = \sum_i (2l_i(u,3)I_i(u) + I_i(u)^2) + \sum_j (2r_j(u,3)O_j(u) + O_j(u)^2) \leq \sum_i (2(l_i(c,1) + T)I_i(u) + I_i(u)^2) + \sum_j (2(r_j(c,1) + T)O_j(u) + O_j(u)^2)$. Similarly, for decrease in potential function, $\mathrm{E}[\Phi(D(u+1,1)) - \Phi(D(u,3)) \mid D(c,1)] \geq \mathrm{E}[\sum_j 4r_j(u,1) - 6n \mid D(c,1)] \geq \sum_j 4(r_j(c,1) - T) - 6n$.

Thus, substituting the bound of expected number of arrivals over the interval, we can show that the expected change in potential function

$$\begin{aligned} &\mathrm{E}[\Phi(D(c+T,1)) - \Phi(D(c,1)) \mid D(c,1)] \\ &= -4T\epsilon \textstyle\sum_j r_j(c,1) + K \end{aligned} \tag{2}$$

where $K = nT(T^2 + 2T + 6)$. Therefore, for some $\delta > 0$, if $\Phi(D(c,1)) > 2(\frac{K+\delta}{4T\epsilon})^2$, then $\sum_j r_j > (K+\delta)/4T\epsilon$ (Lemma 4.3.2). Thus,

$$\mathrm{E}[\Phi(D(c+T,1)) - \Phi(D(c,1)) \mid D(c,1)] < -\delta.$$

Thus, it follows from Foster's criterion, that the queues will infinitely often return to the state where $\Phi(D(t)) \leq 2(\frac{K+\delta}{4T\epsilon})^2$. Hence, WRM achieve 100% throughput. ■

Theorem 4.3.5 If the arrival process is (T,ϵ)-admissible and has a stationary distribution then in the stationary state the expected queue length will be $\frac{9n}{4\epsilon}$.

Proof: Recall from (2) that, $\mathrm{E}[\Phi(D(c+T,1)) - \Phi(D(c,1)) \mid D(c,1)] \leq -4T\epsilon \sum_j r_j(c,1) + K$. Now taking another expectation over $D(c,1)$ we get $\mathrm{E}[\Phi(D(c+T,1))] - \mathrm{E}[\Phi(D(c,1))] \leq -4T\epsilon \mathrm{E}[\sum_j r_j(c,1)] + K$.

But in the stationary state, $\mathrm{E}[\Phi(D(c+T,1))] = \mathrm{E}[\Phi(D(c,1))]$, hence we get $\mathrm{E}[\sum_j r_j(c,1)] \leq \frac{K}{4T\epsilon}$

However, in the stationary state, if an arrival process is (T,ϵ)-admissible then it is also $(1,\epsilon)$-admissible. Thus, substituting $T = 1$, we get $K = 9n$ and $\mathrm{E}[\sum_j r_j(c,1)] \leq \frac{9n}{4\epsilon}$

■

The above result assumes stationarity, hence it does not make sense for adversarial arrival patterns, however, if the arrival process is Markovian then we can bound the expected queue lengths.

4.4 Implementing WRM

The implementation of WRM is fairly simple. Since iSLIP has been implemented in high-speed switches, we will compare the complexity of implementing WRM with that of iSLIP [31]. First of all, since WRM is not iterative, we do not need the extra circuitry used in iSLIP that controls the iterations. Furthermore, we avoid the initial expensive communication phase of iSLIP where each input is required to send a message to each output that it has a packet for. Potentially n^2 messages can be exchanged.

The request step of WRM consists of each input i choosing one of the outputs with probability w_{ij}. This can be achieved by selecting one packet uniformly from all the packets queued at the input and then choosing the corresponding output to send a request message. As a result the probability of choosing a particular output j would be w_{ij}. Specifically we do not need to perform any arithmetic to do this. One of the criticisms of PIM was that it requires random numbers, which were deemed to be expensive to generate. However, this criticism is unwarranted; there are fast and effective random number generators that can be made from a few XOR gates and a 64-bit shift register [44].

Chalasani did an extensive simulation of WRM in [7]. The studies show WRM to be stable for different stochastic and adversarial arrival patterns.

Chapter 5

Summary

We have presented two architectures and associated parallel scheduling algorithms for packet switching.

5.1 SMS architecture

In dissertation we have presented several results on practical routers for output-queued switches based on the SMS architecture. We have presented a pipelined RiPSS, a randomized scheduling algorithm for SMS architecture. Along with theoretical analysis we present extensive numerical results for RiPSS.

The numerical results are interesting for not only this problem, but from a philosophical point of view as well. Most algorithms or heuristics are either supported by an asymptotic (big-O) analysis or simulation results. Both approaches have limitations. Unless the system is simulated for all possible input combinations we cannot guarantee performance based on simulation results. While on the other hand an asymptotic analysis hides constants, thus it is difficult to get an idea about exact numerical running times. The numerical analysis gives a designer concrete numbers to work with for a given switch size.

We have presented a series of pipelined randomized parallel schedulers, PRiPSS.

We have theoretically analyzed these algorithms and presented simulation studies on its performance. The analysis of PRiPSS algorithms are interesting both from theoretical as well as practical point of view. Use of pipelining in the context of a scheduler that gives only probabilistic guarantees was an interesting problem. Usually, correctness of a pipelined algorithm is proved by showing that if an invariant holds for a given cycle, then it holds for the next cycle as well. However, if the invariant is true only with high-probability then the argument does not work. We used the self-stabilizing property of the algorithm to show that the algorithms work.

Our results for RiPSS and PRiPSS-v3 are very encouraging. For switches with up to N= 4,096 input ports, RiPSS placed all incoming packets in compatible memory banks using just 3 rounds in 99.9% of the cycles even when the number of memory banks M was only 1.6N. We have shown that under adversarial conditions, no placement is possible unless $M \geq 2N - 1$, and there exist arrival sequences for which no randomized scheduler can place packets more than half the time unless $M \geq 9N/8$. The fact that RiPSS places all packets under Bernoulli arrivals in just 3 rounds in 99.9% of the cycles when $M \geq 1.6N$ is encouraging.

The effective use of available memory by RiPSS relative to the output-queued switch it simulated is impressive. For both Bernoulli arrival and bursty traffic, most of the memory banks in the SMS switch using RiPSS had load very close to the average load in most cycles. In practical terms, this means that if one uses RiPSS in an SMS architecture, the total amount of buffer space required needs to be only slightly larger than the total number of packets that need to be buffered.

In our simulations with the pipelined scheduler PRiPSS using Bernoulli arrivals, for switches with N up to 2,048 and $M = 1.6N$, a two-stage pipeline with two rounds of communication per stage sufficed to place all packets in memory in every cycle, without exception. When $N = 4,096$ we needed 3 stages in the pipeline, with each stage continuing to have 2 rounds. In view of the extremely small growth rate of the $\log^*$ function, we expect

that PRiPSS will continue to need only 3 stages for much larger values of N.

PRiPSS is superior to RiPSS in terms of throughput. The cycle time of PRiPSS has to be only long enough to accommodate 2 rounds of communication, hence the cycle time can be short, thereby increasing the number of packets processed per second. However, PRiPSS does have the extra overhead of buffering D packets at each input and of computing active packets. If latency is more important, and a small drop rate is tolerable, then RiPSS is better than PRiPSS since for uniform Bernoulli traffic it placed all packets in just 3 rounds in 99.9% of the cycles over the entire range of parameters we considered; in contrast PRiPSS needed 2 stages of 2 rounds each for N up to 2,048, and needed 3 stages of 2 rounds each for $N = 4,096$.

While the choice of RiPSS and PRiPSS may differ with the primary consideration for efficiency, our results indicate that both schedulers perform far better than any other scheduler proposed in the literature for output-queuing, whether implemented directly, or emulated through SMS. In our experiments we also saw that memory utilization in RiPSS is excellent. Even though we did not explicitly measure memory balance for PRiPSS, we expect it to be similar, since the placement method in both RiPSS and PRiPSS is the same.

5.2 CIOQ architecture

Our work on CIOQ architecture was motivated by the objective of designing an extremely simple scheduling algorithm at the expense of minimal speed-ratio. With the WRM scheduler we have been able to achieve that goal. To reiterate our major finding, WRM scheduling algorithm is simpler than all other existing schedulers for IQ switches, schedules variable length packets, and achieves 100% throughput with speed-ratio of 2 for a very general class of admissible traffic. Furthermore, if the order of packets is not important, our scheduler does not even require use of VOQs. Once can just pick a random packet from the input queue and send it.

One question that remains to be answered is that whether a speed-ratio of 2 is abso-

lutely necessary. It is straightforward to see via a "balls-in-bins" argument that a speed-ratio of $\frac{1}{1-1/e} > 1.58$ is necessary for our scheduling algorithm to achieve 100% throughput for the case where adversary manipulates the arrival process to keep all the d_{ij}s equal. Furthermore in the derivation of Equation 1, we use Jensen's inequality and we know that Jensen's inequality is not strict if the corresponding random variable (o_{ij}) does not take a constant value. Thus, our analysis is not tight. It remains an open question as to what minimum speed-ratio is needed for WRM to be stable.

5.3 Future work

In this dissertation we have used crossbar as an interconnection network for transferring packets. As the number of ports increases it becomes more appealing to use interconnection networks with less number of links. A crossbar has $\Theta(N^2)$ links while a Benes network has only $\Theta(N \log N)$ links and a Torus network has $\Theta(N)$ links. However, these interconnection networks are not as powerful as a crossbar. A Benes network can realize any permutation, however, given a permutation, the task of programming the network to realize that permutation is computationally expensive. Thus often a simple randomized algorithm is used that may drop some packets. A Torus network cannot even realize all permutations.

Thus hierarchical interconnection networks bring their own complexity in the system. Often this is solved by use of simple randomized algorithms at the cost of some loss in bandwidth. It would be interesting to study, if the scheduling algorithm that decides which set of packets are transfered in a given cycle and the algorithm that decides the route in the network can have any meaningful interaction such that a combined scheduler yields better throughput.

One interesting idea would be to consider information dispersal in such a setting [25]. The basic idea is to encode a packet into k smaller data blocks such that if at least $l < k$ blocks make it to the output then the packet can be reconstructed. Using such a scheme the blocks are randomly routed through the switching fabric. As randomized rout-

ing is used, all the links in the network are used evenly. This results in good utilization of available bandwidth. If the loss rate is low enough, for any packet, with high probability, enough number of blocks will reach the output such that it can be reconstructed.

Appendix A

RiPSS

Here we give detailed proof of one the key lemmas about RiPSS that gurantees progress in each round.

Lemma 2.4.2 If there are k unmatched inputs and M memories at the beginning of any round and each input can be matched to at least $\epsilon N + \lceil k/s \rceil$ memories, then the expected number of unmatched inputs at the end of that round is at most $ke^{-(1/s+\epsilon N/k)}$. Furthermore the probability that number of unmatched inputs exceed its mean by more than $\sqrt{2M \log M}$ is at most $\frac{1}{M}$.

Proof: First we will bound the expectation. Let $\nu(m)$ be the set of unmatched inputs that can be matched to memory m and let $\eta(i)$ be the set of unmatched memories that can be matched to input i. Clearly $|\nu(m)| \leq k$ and $|\eta(i)| \geq \epsilon N + k/s$.

Let C_m be the index of the input to which memory i sends a request. Thus $\Pr[C_m = j] = 1/|\nu(m)|$ if $j \in \nu(m)$ and 0 otherwise. Let $\mathbf{C} = (C_1, C_2 \ldots, C_M)$ and define the random variable $X_i(\mathbf{C})$ to be 1 if $\forall j.\, (C_j \neq i)$ and 0 otherwise. Informally $X_i(\mathbf{C})$ indicates that input i did not get a request from any of the memories. Since an input is matched if and only if it gets a request from at least one of the memories, $X_i(\mathbf{C}) = 1$ implies input i did not get a match in that round. Let $X(\mathbf{C}) = \sum_i X_i(\mathbf{C})$ be the total number of unmatched

inputs at the end of the round. It is not difficult to see that $\mathrm{E}(X(\mathbf{C})) \leq ke^{-(1/s+\epsilon N/k)}$. Now we will use Azuma's inequality [1] to bound the probability of deviation. Let us define a sequence of random variables Y_0 through Y_M as follows,

$$Y_m(\mathbf{C}) = \mathrm{E}(X(\mathbf{C})|C_1, C_2, \ldots, C_{m-1}).$$

In particular, $Y_0(\mathbf{C})$ is equal to the constant $\mathrm{E}(X(\mathbf{C}))$ and $Y_M(\mathbf{C})$ is identical to $X(\mathbf{C})$. Since $\mathrm{E}(Y_m|Y_{m-1}) = Y_{m-1}$ the sequence of random variables Y_m is a martingale. Furthermore if $\mathbf{C}$ and $\mathbf{C}'$ differ in choice of only one memory then that memory could choose at most one input that was not chosen by any other memory. Thus the difference in number of unmatched inputs can be at most one. Hence by Azuma's inequality we have $\Pr\left[X(\mathbf{C}) > \mathrm{E}(X(\mathbf{C})) + \sqrt{2M \log M}\right] < \frac{1}{M}$. ■

Appendix B

PRiPSS-v3

We now give a detailed analysis of PRiPSS-v3 for the case when $\epsilon > 0$ is an arbitrarily small constant.

It is interesting to note that $\log_b^* N$ is not defined for all values of N, if $b \leq e^{1/e}$. In fact, if $b \leq e^{1/e}$ then $(b \uparrow\uparrow i) \leq e$ for any value of i. Thus, we cannot simply repeat the analysis in Section 4.2 with $b = e^{\frac{(1-\delta)\epsilon}{2}}$.

Recall that $z_i = \frac{N}{2^{i+1}(b\uparrow\uparrow i)}$. We will set $b = 2$ for this analysis. Let D be the smallest integer such that $z_D \leq \sqrt{N}$. Clearly $D = O(\log^* N)$. Let $Q_i(T,t)$ be the set of input ports that have i unmatched packets at the start of t-th iteration of the *pipelined matching procedure* in cycle T, and let $q_i(T,t) = |Q_i(T,t)|$. Let $s_i(T,t) = \sum_{k=i}^{D} q_i(T,t)$.

We define a series of predicates $\Lambda_0(T), \ldots, \Lambda_D(T)$. Predicate $\Lambda_j(T)$ is defined to be true iff for all $i \leq D$, $s_i(T,0) \leq z_{i-j}$, where $z_i = N$ if $i \leq 0$.

Theorem B.0.1 There exists a suitable constant ω such that if each stage executes ω iterations of *pipelined matching procedure* then $\Lambda_0(T)$ implies $\Lambda_0(T+1)$ w.h.p. in N.

In order to prove the above theorem we will first need the following lemma.

Lemma B.0.2 The number of unmatched inputs in the set $Q_i(T,t)$ after a execution of single iteration of pipelined matching procedure is no more than, $q_i(T,t)e^{\epsilon N/2^{i+1}q_i(T,t))}/\alpha$,

w.h.p. in N, where $\alpha > 0$ is a constant independent of N.

Proof: First we bound the expectation. Let j be an input in $Q_i(T,t)$. Let $\eta(j)$ be the set of unmatched memories that can be matched to input j. Clearly $|\eta(j)| \geq \epsilon N + q_i(T,t)$.

Let C_m be the index of the input to which memory m sends a request. Thus, if $m \in |\eta(j)|$, then $\Pr[C_m = j] = 1/(2^{i+1} \cdot q_i(T,t))$. Let $\mathbf{C} = (C_1, C_2 \ldots, C_M)$ and define the random variable $X_j(\mathbf{C})$ to be 1 if $\forall m.\,(C_m \neq j)$ and 0 otherwise. Informally $X_j(\mathbf{C})$ indicates that input j did not get a request from any of the memories. Since an input is matched if and only if it gets a request from at least one of the memories, $X_j(\mathbf{C}) = 1$ implies input j did not get a match in that round. Let $X(\mathbf{C}) = \sum_{j \in Q_i(T,t)} X_i(\mathbf{C})$ be the total number of unmatched inputs in $Q_i(T,t)$ at the end of the round. Then,

$$\begin{aligned} \mathrm{E}(X(\mathbf{C})) &= q_i(T,t)(1 - 1/2^{i+1} q_i(T,t))^{(\epsilon N + q_i(T,t))} \\ &\leq q_i(T,t) e^{-(1+\epsilon N/2^{i+1} q_i(T,t))}. \end{aligned}$$

Thus, defining a martingale and using Azuma's inequality we can say that, w.h.p. in N, number of unmatched inputs in the set $q_i(T,t)$ would be no more than $q_i(T,t)e^{-(\epsilon N/2^{i+1} q_i(T,t))}/\alpha$, where α is a suitable constant. ∎

Lemma B.0.3 If $s_{i+1}(T,t) \leq a$ and $s_i(T,t) \leq a + b$ then, w.h.p. in N, we must have $s_i(T,t+1) \leq a + be^{-N\epsilon/(2^{i+1}b)}/\alpha$.

Proof: Let $q_i(T,t) = x$ and $s_{i+1}(T,t) = y$. If $x \leq \sqrt{N}$ then at the end of that iteration, w.h.p. in N, all the inputs in $Q_i(T,t)$ will be matched. Otherwise, we will have at most $xe^{-N\epsilon/(x2^{i+1})}/\alpha$ inputs with i unmatched inputs that were also in $Q_i(T,t)$ (from Lemma B.0.2). Let δ be the number of inputs that got matched in $Q_{i+1}(T,t)$ thus, $q_i(T,t+1) \leq xe^{-N\epsilon/(2^{i+1}x)}/\alpha + \delta$ and $s_{i+1}(T,t+1) \leq y - \delta$. Therefore, $s_i(T,t+1) = s_{i+1}(T,t+1) + q_i(T,t+1) \leq y + xe^{-N\epsilon/(2^{i+1}x)}/\alpha$. Thus,

$$s_i(T,t+1) \leq \max_{y \leq a,\ x+y \leq a+b} (y + \frac{xe^{-N\epsilon/(2^{i+1}x)}}{\alpha}).$$

It is straightforward to show that the function on the R.H.S. achieves its maxima at $y = a$ and $x = b$. Substituting that we get the desired result. ∎

Substituting $a = z_{i+1}$, $b = z_i - z_{i+1}$ and $t = 0$ in the above lemma we get $s_i(T,1) \leq z_{i+1} + \frac{z_i - z_{i+1}}{\alpha}$. Since $z_{i+1} \leq z_i/2$ we get $s_i(T,1) \leq \beta z_i$, where $\beta = 1/2 + 1/2\alpha < 1$. Similarly $s_{i+1}(T,1) \leq \beta z_{i+1}$. Thus, applying this argument repeatedly we get $s_i(T,f) \leq \beta^f z_i$. Let g be a constant such that $\beta^g \leq \min(1/2, \epsilon/\ln 2)$. Thus, $s_i(T,g) \leq z_i\beta^g$ and $s_{i+1}(T,g) \leq z_{i+1}\beta^g$.

Substituting $a = z_{i+1}\beta^g$ and $b = z_i\beta^g$ and $t = g$ in Lemma B.0.3 for the next iteration it is not difficult to show that

$$s_i(T, g+1) \leq \beta^g \left(z_{i+1} + z_i e^{-\frac{\epsilon N}{\beta^g 2^i z_i}} \right) \leq z_{i+1}.$$

Thus, if we set $\omega \geq g + 1$, We have $s_{i-1}(T,\omega) \leq z_i$. Since at most one packet arrives in a cycle, $s_i(T+1,0) \leq s_{i-1}(T,\omega) \leq z_i$. Hence, $\Lambda_0(T+1)$ holds with high probability in N.

Lemma B.0.4 If $\Lambda_0(T)$ is true, w.h.p. in N, all packets that arrived in cycle $T - D$ will be matched at the end of cycle T.

Proof: From the definition of $\Lambda_0(T)$ we get $q_D(T,0) = s_D(T,0) \leq \sqrt{N}$. Thus, w.h.p. in N, all the inputs in $Q_D(T,0)$ are matched in the first iteration of *pipelined matching procedure*. Thus, $q_D(T,1) = 0$, i.e., no input has D unmatched packets. Thus, all the packets that arrived $T - D$ cycles earlier are matched. ∎

Since $\Lambda_0(0)$ is trivially true, by Theorem B.0.1 we can argue inductively that $\Lambda_0(T)$ is true for $T = O(N)$. However, as T grows large, the probability that $\Lambda_0(T)$ will continue to be true becomes small and then we can no longer guaranty that all the packets that arrived in cycle $T - D$ will be matched at the end of cycle T. However, if we set $\omega \geq 2(g+1)$ our algorithm becomes "self-stabilizing" , i.e., if $\Lambda_0(T)$ becomes false for some T, then within D cycles the input queues get back to a state where the predicate Λ_0 is true.

Note that $\Lambda_j(T)$ implies $\Lambda_k(T)$ if $k \geq j$.

Theorem B.0.5 If $j > 0$ and $\Lambda_j(T)$ is true then, w.h.p, $\Lambda_{j-1}(T+1)$ is true.

Proof: Recall that in the proof of Theorem B.0.1 we proved that if $s_i(T,0) \leq z_i$ then $s_i(T, g+1) \leq z_{i+1}$. Using a similar argument if $s_i(T,0) \leq z_{i-j}$ then $s_i(T,(g+1)) \leq z_{i-j+1}$. If we apply another $g+1$ iterations we get $s_i(T, 2(g+1)) \leq z_{i-j+2}$. Thus, setting $\omega = 2(g+1)$, we get $s_i(T+1,0) \leq s_{i-1}(T, 2(g+1)) \leq z_{i-j+1}$. Hence, $\Lambda_{j-1}(T+1)$ holds. ■

Since $\Lambda_D(T)$ is trivially true, in D steps we get back to a state where $\Lambda_0(T)$ is true. This establishes the self-stabilizing feature of PRiPSS-v3.

Bibliography

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. John Wiley, 2000.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High-speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, November 1993.

[3] A. Aziz, A. Prakash, and V. Ramachandran. A near optimal scheduler for switch-memory-switch routers. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 343–352, San Diego, CA, June 2003.

[4] R. Barker, P. Massiglia, and L. Krantz. *Storage Area Networking Essentials*. McGraw-Hill, 2001.

[5] A. J. Blanksby and C. J. Howland. A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Decoder. *IEEE Journal of Solid-State Circuits*, 37:404–412, March 2002.

[6] P. Brèmaud. *Markov Chains*. Springer-Verlag, 1999.

[7] S. Chalasani. *A Performance Comparison of Three Scheduling Algorithms for Input-queued Switches*. Masters Report, Department of Electrical and Computer Engineering, University of Texas at Austin, May 2003.

[8] C.-S. Chang, D.-S. Lee, and Y.-S. Jou. Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering. *Computer Communications*, 2001.

[9] C.-S. Chang, D.-S. Lee, and C.-M. Lien. Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering. *Computer Communications*, 2001.

[10] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. In *IEEE Infocom*, volume 3, pages 1169–1178, 1999.

[11] T. M. Cover and J. A. Thomas. *Information Theory*. Wiley, John & Sons, Incorporated, 1991.

[12] A. Czumaj, F. Meyer auf de Heide, and V. Stemann. Contention resolution in hashing based shared memory simulations. In *SIAM Journal on Computing*, volume 29, pages 1703–1739, 2000.

[13] J. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *IEEE Infocom*, pages 556–564, 2000.

[14] J. Duato. *Interconnection Networks*. Morgan-Kaufmann, 2002.

[15] A. E. Eckberg and T. C. Hou. Effects of output buffer sharing on buffer requirements in an atdm packet switch. In *IEEE Infocom*, 1988.

[16] M. Farley. *Building storage area networks*. McGraw-Hill, 2001.

[17] T. Feder, N. Megiddo, and S. Plotkin. A sublinear parallel algorithm for stable matching. In *Symposium on Discrete Algorithms*, pages 297–308, 1994.

[18] W. Futral. *InfiniBand Architecture: Development and Deployment–A Strategic Guide to Server I/O Solutions*. Intel Press, 2001.

[19] P. Giaccone, B. Prabhakar, and D. Shah. Towards simple, high-performance schedulers for high-aggregate bandwidth switches. In *IEEE Infocom*, New York City, June 2002.

[20] John Hennessy, David Patterson, and David Goldberg. *Computer Architecture: A Quantitative Approach*. Morgan-Kaufmann, third edition, 2002.

[21] A. Israeli and Y. Shiloach. An Improved Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22:57–60, 1986.

[22] S. Iyer, R. Zhang, and N. McKeown. Routers with a single stage of buffering. In *ACM SIGCOMM*, pages 251–264, 2002.

[23] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley, 1997.

[24] I. Keslassy and N. McKeown. Maintaining Packet Order in Two-Stage Switches. In *IEEE Infocom*, 2002.

[25] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan-Kaufmann, 1991.

[26] E. Leonardi, M. Mellia, F. Neri, and M. Marsan. On the stability of input-queued switches with speed-up. *IEEE Transactions on Networking*, 9(1):104–118, February 2001.

[27] Y. Matias and U. Vishkin. Towards a theory of nearly constant time parallel algorithms. In *IEEE Conference on Decision and Control*, pages 318–325, 1991.

[28] N. McKeown. *Scheduling algorithms for input switched queues*. PhD thesis, The University of California at Berkeley, 1995.

[29] N. McKeown. iSLIP: A Scheduling Algorithm for Input-Queued Switches. *IEEE Transactions on Networking*, 7(2):188–201, April 1999.

[30] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *IEEE Infocom*, 1996.

[31] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. The tiny tera: a packet switch core. *IEEE Micro*, 17(1):27–33, January 1997.

[32] M. Mohiyuddin, A. Prakash, A. Aziz, and W. Wolf. Synthesizing Interconnect-Efficient Low Density Parity Check Codes. In *Design Automation Conference*, San Diego, CA, June 2004.

[33] Juniper Networks. High speed switching device. US Patent 5,905,726, 1999.

[34] R. Panigrahy, A. Prakash, A. Nemat, and A. Aziz. Weighted random matching: a simple scheduling algorithm for achieving 100% throughput. In *IEEE Workshop High Performance Switching and Routing*, pages 111–115, April 2004.

[35] L. Peterson and B. Davie. *Computer Networks*. Morgan-Kaufmann, 2000.

[36] A. Prakash. Lower bound on performance of RM. manuscript, February 2003.

[37] A. Prakash and A. Aziz. OC-3072 Packet Classification Using BDDs and Pipelined SRAMs. In *Hot Interconnects*, Stanford University, CA, August 2001.

[38] A. Prakash and A. Aziz. A middle ground between CAMs and DAGs for high-speed packet classification. In *Hot Interconnects*, Stanford University, CA, August 2002.

[39] A. Prakash, A. Aziz, and V. Ramachandran. Randomized parallel schedulers for switch-memory-switch routers: Analysis and numerical studies. In *IEEE Infocom*, Hong Kong, March 2004.

[40] A. Prakash, R. Kotla, T. Mandal, and A. Aziz. A reconfigurable architecture and associated synthesis methodology for high speed packet classification. In *Proceedings of the International Workshop on Logic Synthesis*, 2002.

[41] A. Prakash, R. Kotla, T. Mandal, and A. Aziz. A high-performance architecture and bdd-based synthesis methodology for packet classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 83(10), June 2003.

[42] A. Prakash, S. Sharif, and A. Aziz. An $O(\lg^2 n)$ algorithm for output queuing. In *IEEE Infocom*, pages 1623–1629, Ney York, NY, 2002.

[43] R. Ramaswami and K. Sivarajan. *Optical networks: a practical perspective*. Morgan-Kaufmann, 2001.

[44] G. Solomon. *Shift Register Sequences*. Aegean Park Press, 1982.

[45] T. Stern and K. Bala. *Multiwavelength optical networks: a layered approach*. Prentice-Hall, 1999.

[46] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *IEEE Infocom*, pages 533–539, 1998.

[47] J. van Lint and R. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.

[48] A. Wilson, J. Schade, and R. Thornburg. *Introduction to PCI Express*. Intel Press, 2002.

[49] K. Yoshigoe and K. Christensen. A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar. In *IEEE Workshop High Performance Switching and Routing*, pages 271–275, Dallas, TX, USA, May 2001.

Vita

Amit Prakash was born in Muzaffarpur, India on March 1, 1977. He joined the Indian Institute of Technology, Kanpur in 1994 where he received the Bachelor of Technology degree in Electrical Engineering in May 1998. He worked in Synopsys India as a research and development engineer for about a year in the hardware-software co-verification team. In June 1999 he joined the graduate program in the Department of Electrical and Computer Engineering at the University of Texas at Austin. Currently he is working under the supervision of Dr. Adnan Aziz in the area of algorithms and VLSI architectures for high speed networking.

Permanent Address: Prakash Kutir, Daudpur Kothi
MIT, Muzaffarpur, Bihar - 842003 India

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.